WebSphere® Application Server-Network Deployment, Version 6

Troubleshooting and support

Note

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

Compilation date: December 8, 2004

Contents

How to send your comments

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

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

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

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

Chapter 1. Overview and new features for troubleshooting

This topic summarizes the contents and organization of the troubleshooting documentation, including links to conceptual overviews and descriptions of new features.

- “Overview: Troubleshooting”
- “What is new for troubleshooters” on page 2

Sections in the troubleshooting documentation:

Chapter 3, “Debugging applications,” on page 61
This topic describes how to use Java logging for generating your application 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.

Chapter 4, “Adding logging and tracing to your application,” on page 63
This topic describes how to use Java logging for generating your application 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.

Chapter 5, “Diagnosing problems (using diagnosis tools),” on page 121
This topic describes skills and tools for collecting, displaying, and diagnosing troubleshooting data. It includes dumps, messages, logging, tracing, and debugging.

Troubleshooting by task
This topic summarizes how to detect, confirm, and solve the problem, based on what you were trying to do when the problem occurred.

Troubleshooting WebSphere applications
For various components that might have failed, this topic summarizes how to detect, confirm, and solve the resulting problem. The instructions are specific to various application types. For example, you can focus on troubleshooting access problems for an enterprise bean application, or a problem that displayed as soon as you enabled security.

Overview: Troubleshooting

This topic provides links to conceptual overviews of troubleshooting problems involving deployed applications and the application serving environment.

“What is new for troubleshooters” on page 2

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

Chapter 5, “Diagnosing problems (using diagnosis tools),” on page 121

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

Presentations from Education on Demand

- Where to start with problem determination
- Serviceability enhancements

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 cryptic 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? See the topic on searching knowledge bases.
- Is there a fix you can download? See the topic on getting fixes.
- You can use the WebSphere Application Server Troubleshooting Guide to 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 5, “Diagnosing problems (using diagnosis tools),” on page 121.

What is new for troubleshooters

This topic highlights what is new or changed in Version 6 for users who are going to use the troubleshooting tools and serviceability features.

The biggest change in troubleshooting support and serviceability is increased ability to automatically detect and recover from problems. Many serviceability improvements are described in the Education on Demand presentation:Serviceability enhancements.

The troubleshooting section has been revamped and expanded to include extensive support information, including the ability to search live, Web-based support resources by using the customized query fields in the ″Web search″ page.

New troubleshooting technology on the Support site

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

Improved message text, new message prefixes

Messages for key product components used during installation, migration, and initial configuration have been improved. Additional components have messages now. Message prefixes have changed. The message reference provides a mapping of Version 5.1.x prefixes to Version 6.0.x prefixes.

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 “Logging and tracing with Java logging” on page 63.

JRAS is deprecated

The JRAS API is deprecated. Users are directed to use the JSR47 logging infrastructure instead. See Deprecated features in Version 6.0 for more information about this and other deprecated items.

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 81.

Thread names can be included in logs

Thread name has been added to the Advanced log format and Log Analyzer format. 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.

Troubleshooting by component

Select a component from the list.

Web module or application server dies or hangs

If an application server dies (its process spontaneously closes), or freezes (its Web modules stop responding to new requests):

- Isolate the problem by installing Web modules on different servers, if possible.
- Read the Monitoring performance with Tivoli performance viewer (formerly resource analyzer) topic. You can use the 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.
- 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* > 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.

- If either the performance viewer or verbose garbage collection output indicates that the application server is running out of memory, one of the following problems might be present:
  - There is a memory leak in application code that you must address. To pinpoint the cause of a memory leak, enable the **RunHProf** function in the Servers > Application Servers > server_name > Process Definition > Java Virtual Machine pane of the problem application server:
    - In the same JVM pane, set the **HProf Arguments** field to a value similar to `depth=20,file=heapdmp.txt`. This value shows exception stacks to a maximum of 20 levels, and saves the heapdump output to the *`install_root`*`/bin/heapdmp.txt` file.
    - Save the settings.
    - Stop and restart the application server.
    - Reenact the scenario or access the resource that causes the hang or crash, if possible. Stop the application server. If this is not possible, wait until the hang or crash happens again and stop the application server.
    - Examine the file into which the heapdump was saved. For example, examine the *`install_root`*`/bin/heapdmp.txt` file:
      - Search for the string, ″SITES BEGIN″. This finds the location of a list of Java objects in memory, which shows the amount of memory allocated to the objects.
      - The list of Java objects occurs each time there was a memory allocation in the JVM. There is a record of what type of object the memory instantiated and an identifier of a trace stack, listed elsewhere in the dump, that shows the Java method that made the allocation.
      - The list of Java object is in descending order by number of bytes allocated. Depending on the nature of the leak, the problem class should show up near the top of the list, but this is not always the case. Look throughout the list for large amounts of memory or frequent instances of the same class being instantiated. In the latter case, use the ID in the `trace stack` column to identify allocations occurring repeatedly in the same class and method.
      - Examine the source code indicated in the related trace stacks for the possibility of memory leaks.
  - The default maximum heap size of the application server needs to be increased.
- If an application server spontaneously dies, look for a Java thread dump file. The JVM creates the file in the product directory structure, with a name like `javacore[number].txt`.
- 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 installation root directory with a name like `javacore.`*`date.time.id`*`.txt`.
- Browse the thread dump for clues:
  - If the JVM creates the thread dump as it closes (the thread dump is not manually forced), there might be ″error″ or ″exception information″ strings at the beginning of the file. These strings indicate the thread that caused the application server to die.
  - The thread dump contains a snapshot of each thread in the process, starting in the section labeled ″Full thread dump.″
    - Look for threads with a description that contains ″state:R″. Such threads are active and running when the dump is forced, or the process exited.

- 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.

If these steps do not fix your problem, search to see if the problem is known and documented, using the methods identified in the available online support (hints and tips, technotes, and fixes) topic. If you find that your problem is not known, contact IBM support to report it.

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.

Web Container troubleshooting tips

If you are having problems starting a Web module, or accessing resources within a particular Web module:

- View the JVM logs and process logs for the application server which hosts the problem Web modules, and look for messages in the JVM output file which indicate that the web module has started successfully. You should see messages similar to the following:

```
WebContainer  A SRVE0161I: IBM WebSphere Application Server - Web Container.
Copyright IBM Corp. 1998-2002
WebContainer  A SRVE0169I: Loading Web Module: [module_name]
ApplicationMg A WSVR0221I: Application started: [application_name]
HttpTransport A SRVE0171I: Transport http is listening on port [port_number]
[server_name] open for e-business in [install_root]/log/[server_name]/SystemOut.log
```

- For specific problems that can cause servlets, HTML files, and JavaServer Pages (JSP) files not to be served, seeWeb resource (JSP file, servlet, HTML file, image) does not display .
- Use the Log Analyzer tool to browse the service log (activity.log) file for clues.
- For a detailed trace of the run-time behavior of the Web container, enable trace for the component com.ibm.ws.webcontainer.*.

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

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.

JSP source code shown by the Web server

Problem

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 JSP source file as plain text.

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.

JSP engine troubleshooting tips

If you are having difficulty using the JSP engine, try these steps:

1. Determine whether other resources such as `.html` files or servlets are being requested and displayed correctly. If they are not, the problem probably lies at a deeper level, such as with the HTTP server.
2. If other resources are being displayed correctly, determine whether the JSP 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 .
   - Open the Log analyzer on the service log of the server which is hosting the JavaServer Pages file you are trying to access and use it to browse error and warning messages.
3. If the JSP engine has started normally, the problem may be with the JSP file itself.
   - 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.

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.

HTTP session manager troubleshooting tips

If you are having problems creating or using HTTP sessions with your Web application hosted by WebSphere Application Server, here are some steps to take:

- See HTTP session aren't getting created or are getting dropped to see if your specific problem is discussed.
- View the JVM logs for the application server which hosts the problem application:
  - first, look at messages written while each application is starting. They will be written between the following two messages:

    ```
    Starting application: application
    ..................
    Application started: application
    ```
  - Within this block, look for any errors or exceptions containing a package name of com.ibm.ws.webcontainer.httpsession. If none are found, this is an indication that the session manager started successfully.
  - Error ″**SRVE0054E: An error occurred while loading session context and Web application**″ indicates that SessionManager didn't start properly for a given application.
  - Look within the logs for any Session Manager related messages. These messages will be in the format SESNxxxxE and SESNxxxxW for errors and warnings, respectively, where xxxx is a number identifying the precise error. Look up the extended error definitions in the Session Manager message table.
- Use the Log Analyzer tool to browse the service log (activity.log) file for clues.
- See Best practices for using HTTP Sessions.
- To dynamically view the number of sessions as a Web application is running, enable performance monitoring for HTTP sessions. This will give you an indication as to whether sessions are actually being created.
- To learn how to view the http session counters as the application runs, see Monitoring performance with Tivoli Performance Viewer (formerly Resource Analyzer).
- Alternatively, a special servlet can be invoked that displays the current configuration and statistics related to session tracking. This servlet has all the counters that are in performance monitor tool and has some additional counters.
  - Servlet name: **com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug**.
  - It can be invoked from any web module which is enabled to serve by class name. For example, using default_app, **http://localhost:9080/servlet/com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug**.
  - If you are viewing the module via the serve-by-class-name feature, be aware that it may be viewable by anyone who can view the application. You may wish to map a specific, secured URL to the servlet instead and disable the serve-servlets-by-classname feature.
- Enable tracing for the HTTP Session Manager component:
  - Use the trace specification **com.ibm.ws.webcontainer.httpsession.*=all=enabled**. Follow the instructions for dumping and browsing the trace output to narrow the origin of the problem.
  - If you are using persistent sessions based on memory replication, also enable trace for **com.ibm.ws.drs.***.
- If you are using **database-based persistent sessions**, look for problems related to the **data source** the Session Manager relies on to keep session state information. For details on diagnosing database related problems see Errors accessing a datasource or connection pool

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.

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

Note: To view and update the Session Manager settings discussed here, use the administrative console. Select the application server that hosts the problem application, then under **Additional properties**, select **Web Container**, then **Session manager**.

What kind of problem are you having?
- HTTP Sessions are not getting created, or are lost between requests.
- HTTP Sessions are not persistent (session data lost when application server restarts, or not shared across cluster).
- Session is shared across multiple browsers on same client machine.
- Session is not getting invalidated immediately after specified Session timeout interval.
- Unwanted sessions are being created by jsps.
- Session data intended for one client is seen by another client

If your problem is not described here, or none of these steps fixes the problem:
- Review Troubleshooting the HTTP Session Manager for general steps on debugging Session-manager related problems.
- Review Managing HTTP sessions for information on how to configure the Session manager, and best practices for using it.
- Check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes).
- If you don't find your problem listed there contact IBM support.

HTTP Sessions are not getting created, or are lost between requests

By default, the Session Manager uses cookies to store the session ID on the client between requests. Unless you intend to avoid cookie-based session tracking, ensure that cookies are flowing between WebSphere Application Server and the browser:
- Make sure the **Enable cookies** checkbox is checked under the **Session tracking Mechanism** property.
- Make sure cookies are enabled on the browser you are testing from or from which your users are accessing the application.
- Check the Cookie domain specified on the SessionManager (to view the or update the cookie settings, in the **Session tracking mechanism->enable cookies** property, click **Modify**).
  - For example, if the cookie domain is set as ″.myCom.com″, resources should be accessed using that domain name, e.g. http://www.myCom.com/myapp/servlet/sessionservlet.

  - If the domain property is set, make sure it begins with a dot (.). Certain versions of Netscape do not accept cookies if domain name doesn't start with a dot. Internet Explorer honors the domain with or without a dot. For example, if the domain name is set to *mycom.com*, change it to *.mycom.com* so that both Netscape and Internet Explorer honor the cookie.
- Check the **Cookie path** specified on the SessionManager. Check whether the problem url is hierarchially below the Cookie path specified. If not correct the Cookie path.
- If the `Cookie maximum age` property is set, ensure that the client (browser) machine's date and time is the same as the server's, including the time zone. If the client and the server time difference is over the ″Cookie maximum age″ then every access would be a new session, since the cookie will ″expire″ after the access.
- If you have multiple web modules within an enterprise application that track sessions:
  - If you want to have different session settings among web modules in an enterprise application, ensure that each web module specifies a different cookie name or path, or
  - If Web modules within an enterprise application use a common cookie name and path, ensure that the HTTP session settings, such as Cookie maximum age, are the same for all Web modules. Otherwise cookie 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, excericise your session-using servlet or jsp, then follow the instructions for dumping and browsing the trace output .
  3. Access the session servlet from the browser.
  4. The browser will prompt for the cookie; note the jsessionid.
  5. Reload the servlet, note down the cookie if a new cookie is sent.
  6. Check the session trace and look for the session id and trace the request by the thread. Verify that the session is stable across web requests:
     - Look for **getIHttpsession(...)** which is start of session request.
     - Look for **releaseSesson(..)** which is end of servlet request.
- If you are using URL rewriting instead of cookies:
  - Ensure there are no static HTML pages on your application's navigation path.
  - Ensure that your servlets and jsp files are implementing URL rewriting correctly. For details and an example see Session tracking options.
- If you are using SSL as your session tracking mechanism:
  - Ensure that you have SSL enabled on your IBM HTTP Server or iPlanet HTTP server.
  - Review Session tracking with SSL information.
- If you are in a clustered (multiple node) environment, ensure that you have session persistence enabled.

HTTP Sessions are not persistent

If your HTTP sessions are not persistent, that is session data is lost when the application server restarts or is not shared across the cluster:
- Check the Datasource.
- Check the SessionManager's Persistence Settings properties:
  - If you intend to take advantage of Session Persistence, verify that Persistence is set to **Database**or **Memory to Memory Replication**.
  - If you are using **Database-based persistence**:
    - Check the jndi name of the datasource specified correctly on SessionManager.
    - Specify correct userid and password for accessing the database.

      Note that these settings have to be checked against the properties of an existing Data Source in the admin console. The Session Manager does not automatically create a session database for you.
    - The Datasource should be non-JTA, i.e. non XA enabled.
    - Check the JVM logs for 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.
  – If you are using **memory-based persistence**, available in a network-deployment (multiple application server) configuration only:
    - Review .
    - Review the **Internal Replication Domains properties** of your Session manager.

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 specifed maximum age time.

Session is not getting invalidated immediately after specified Session timeout interval

The SessionManager invalidation process thread runs every *x* seconds to invalidate any invalid sessions, where x is determined based on the Session timeout interval specified in the Session manager properties. For the default value of 30 minutes , *x* is around 300 seconds. In this case, it could take up to 5 minutes (300 seconds) beyond the timeout threshold of 30 minutes for a particular session to become invalidated.

Unwanted sessions are being created by jsps

As required by the JavaServer Page specification, jsps by default perform a request.getSession(true), so that a session is created if none exists for the client. To prevent jsps from creating a new session, set the session scope to **false** in the jsp 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.

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.

Enterprise bean and EJB container troubleshooting tips

If you are having problems starting an EJB container, or encounter error messages or exceptions that appear to be generated on by an EJB container, follow these steps to resolve the problem:

- Browse the relevant log files for clues:

  - Use the Administrative Console to verify that the application server which hosts the container is running.
  - Browse the JVM log files for the application server which hosts the container. Look for the message **server *server_name* open for e-business** in the `SystemOut.log` . If it does not appear, or if you see the message **problems occurred during startup**, browse the `SystemErr.log` for details.
  - Browse the system log files for the application server which hosts the container.
- Use the Log Analyzer tool to browse the service log file for more information.
- Enable tracing for the EJB Container component, by using the following trace specification `EJBContainer=all=enabled`. Follow the instructions for dumping and browsing the trace output to narrow the origin of the problem.

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

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 access an enterprise bean from a servlet, a JSP file, a stand-alone program, or another client

What kind of error are you seeing?

- **javax.naming.NameNotFoundException: Name** *name* **not found in context** ″**local**″ **message** when access is attempted
- **BeanNotReentrantException** is thrown
- **CSITransactionRolledbackException / TransactionRolledbackException** is thrown
- Call fails, Stack trace beginning **EJSContainer E Bean method threw exception [exception_name]** found in JVM log file.
- Call fails, **ObjectNotFoundException or ObjectNotFoundLocalException** when accessing stateful session EJB found in JVM log file.
- Attempt to start CMP EJB module fails with **javax.naming.NameNotFoundException: *dataSourceName***
- **Transaction [tran ID] has timed out after 120 seconds** error accessing EJB.
-
- Symptom: **CNTR0001W: A Stateful SessionBean could not be passivated**
- 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 40
   - “Naming services component troubleshooting tips” on page 39
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“Obtaining help from IBM” on page 155 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 255.″

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 throws 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 throws a NullPointerException, ArrayIndexOutOfBoundsException, or other Java runtime exception, or where a BMP bean encounters a

JDBC error. The resolution is to investigate the enterprise bean code and resolve the underlying exception, or to add the exception to the problem method 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 40.

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: StatefulBeanO
(BeanId(XXX#YYY.jar#ZZZZ, <ThreadID>),
state = PASSIVATING) com.ibm.ejs.container.passivator.StatefulPassivator@<ThreadID>
java.io.NotSerializableException: com.ibm.ws.rsadapter.jdbc.WSJdbcConnection
 at java.io.ObjectOutputStream.outputObject((Compiled Code))
 at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java(Compiled Code))
 at java.io.ObjectOutputStream.outputClassFields((Compiled Code))
 at java.io.ObjectOutputStream.defaultWriteObject((Compiled Code))
 at java.io.ObjectOutputStream.outputObject((Compiled Code))
 at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java(Compiled Code))
 at com.ibm.ejs.container.passivator.StatefulPassivator.passivate((Compiled Code))
```

```
at com.ibm.ejs.container.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.

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.

Universal Discovery, Description, and Integration, Web Service, and SOAP component troubleshooting tips

If you are having problems deploying or executing applications that use WebSphere Application Server Web Services, Universal Discovery, Description, and Integration (UDDI), or SOAP, try these steps:

- Review the troubleshooting documentation for messaging in the information center:
  - WSIF troubleshooting tips
- Investigate the following areas for SOAP-related problems:
  - View the JVM logs for the target application server, and run the Log Analyzer on the server's service log.
  - View the error log of the HTTP server to which the SOAP request is sent.
  - View the run-time behavior of the SOAP component in more detail, by enabling trace for `org.apache.soap.*` and `com.ibm.*.soap*`.
  - Browse the Web site `http://xml.apache.org/soap/` for FAQs and known SOAP issues.

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

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 returned to a client sending a SOAP request

What kind of problem are you seeing?

- SOAPException: faultCode=SOAP-ENV:Client; msg=Error opening socket; java.net.ConnectException: Connection refused: connect
- javax.security.cert.CertPathBuilderException: No end-entity certificate matching the selection criteria could be found.

If none of these errors match the one you see:

- Browse the target application server log files. (*installation_directory*/server_name/SystemErr.log and SystemOut.log for clues. See "Viewing the JVM logs" on page 123 for more information.
- Look up any error or warning messages in the message table.
- See "Universal Discovery, Description, and Integration, Web Service, and SOAP component troubleshooting tips" for more information.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see"Obtaining help from IBM" on page 155.

SOAPException: faultCode=SOAP-ENV:Client; msg=Error opening socket; java.net.ConnectException: Connection refused: connect

The most likely cause of this refused connection is that it was sent to the default port, 80, and an HTTP server is not installed or configured.

To verify this situation, send the message directly to the SOAP port; for example, to `http://`*`hostname`*`:9080`. If the message is sent correctly, there are two ways to resolve the problem:

- Continue specifying port 9080 on SOAP requests.
- If an HTTP server is not installed, install one and the associated plug-in component.
- If an HTTP server is installed:
  - Regenerate the HTTP plug-in configuration in the administrative console by clicking **Environment > Update WebServer Plugin**, and restarting the HTTP server.

– If the problem persists, view the HTTP server access and error logs, as well as the *plugin_install_root*/logs/*web_server_name*/http_plugin.log file for more information.

javax.security.cert.CertPathBuilderException: No end-entity certificate matching the selection criteria could be found

This error usually indicates that new or updated security keys are needed. The security key files are:
- SOAPclient
- SOAPserver
- sslserver.p12

In an installed application, these files are located in the:
install_dir/installedApps/*application_name*.ear/soapsec.war/key/ directory. After replacing these files, you must stop and restart the application.

To replace these files in a SOAP-enabled application that is not yet installed:
- Expand the *application_name*.ear file.
- Expand the soapsec.war file.
- Replace the security key files in thekey/ directory.
- After you replace these files, install the application and restart the server.

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.

JDBC and data source troubleshooting tips

To see whether your specific problem has been addressed, review the Cannot access a data source topic. If you cannot find your problem described there, investigate the problem further by using the following instructions to enable tracing for relevant WebSphere Application Server components.

This topic includes the following sections:
- Trace strings for JDBC data sources
- JDBC trace properties

Trace strings for JDBC data sources

Turn on JDBC tracing by using the following trace strings:
- **com.ibm.ws.database.logwriter** Trace string for databases that use the GenericDataStoreHelper. You can also use this trace string for unsupported databases.
- **com.ibm.ws.db2.logwriter** Trace string for DB2 databases.
- **com.ibm.ws.oracle.logwriter** Trace string for Oracle databases.
- **com.ibm.ws.cloudscape.logwriter** Trace string for Cloudscape databases.
- **com.ibm.ws.informix.logwriter** Trace string for Informix databases.
- **com.ibm.ws.sqlserver.logwriter** Trace string for Microsoft SQL Server databases.
- **com.ibm.ws.sybase.logwriter** Trace string for Sybase databases.

The trace group that includes the trace strings is WAS.database.

JDBC trace properties

Use a back-end database that supports JDBC tracing. Setting trace strings does not result in a trace if the database does not support JDBC tracing. The following databases offer JDBC tracing at this time:
- DB2

- Oracle
- SQL Server

Set the level of trace desired for DB2 Universal database and Oracle as custom properties on the datasource.

- **DB2 Universal JDBC driver provider** Custom properties for DB2 are:
  - **traceLevel** Possible traceLevel values are:
    - TRACE_NONE = 0
    - TRACE_CONNECTION_CALLS = 1
    - TRACE_STATEMENT_CALLS = 2
    - TRACE_RESULT_SET_CALLS = 4
    - TRACE_DRIVER_CONFIGURATION = 16
    - TRACE_CONNECTS = 32
    - TRACE_DRDA_FLOWS = 64
    - TRACE_RESULT_SET_META_DATA = 128
    - TRACE_PARAMETER_META_DATA = 256
    - TRACE_DIAGNOSTICS = 512
    - TRACE_SQLJ = 1024
    - TRACE_ALL = -1

    Note: This trace level provides real data that sets to the PreparedStatement or gets from the ResultSet object.
  - **traceFile** Use this property to integrate the DB2 trace with the WebSphere Application Server trace. If you do not set the value, traces are integrated. Otherwise, DB2 traces are directed to the desired file. You can dynamically enable or disable trace. You can run an application and turn on the DB2 trace if there is a problem. Use the run time trace enablement provided with the Application Server by specifying a trace string of com.ibm.ws.db2.logwriter=all=enabled.
- **Oracle JDBC provider** Custom properties for Oracle are:
  - **oraclelogCategoryMask** Controls the output category. The default is 47, which is (OracleLog.USER_OPER 1 | OracleLog.PROG_ERROR 2 | OracleLog.ERROR 4 | OracleLog.WARNING 8 | OracleLog.DEBUG1 32).

    Possible values are:
    - OracleLog.USER_OPER 1
    - OracleLog.PROG_ERROR 2
    - OracleLog.ERROR 4
    - OracleLog.WARNING 8
    - OracleLog.FUNCTION 16
    - OracleLog.DEBUG1 32
    - OracleLog.SQL_STR 128
  - **oraclelogModuleMask** Controls which modules write debug output. The default is 1, which is OracleLog.MODULE_DRIVER 1.

    Possible values are:
    - OracleLog.MODULE_DRIVER 1,
    - OracleLog.MODULE_DBACCESS 2
  - **oraclelogPrintMask** Controls which information to print with each trace message. The default is 62, which is ([OracleLog.FIELD_OBJECT for 9i / OracleLog.FIELD_CONN for 8i] 32 | OracleLog.FIELD_CATEGORY 16 | OracleLog.FIELD_SUBMOD 8 | OracleLog.FIELD_MODULE 4 | OracleLog.FIELD_TIME 2).

    Possible values are:
    - OracleLog.FIELD_TIME 2
    - OracleLog.FIELD_MODULE 4
    - OracleLog.FIELD_SUBMOD 8
    - OracleLog.FIELD_CATEGORY 16
    - OracleLog.FIELD_OBJECT 32
    - OracleLog.FIELD_THREAD 64

Notes for Oracle JDBC tracing:

1. Oracle 9i requires the use of `classes12_g.zip` to display traces. With Oracle8i, the `classes12_g.zip` is optional.
2. You can dynamically enable or disable trace. You can run an application and turn on the Oracle trace if there is a problem. Use the run-time trace enablement provided with the WebSphere Application Server products, by specifying a trace string of com.ibm.ws.oracle.logwriter=all=enabled.

If JDBC tracing does not provide enough information to isolate and fix your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you do not find your problem listed there, contact IBM Support.

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.

DB2 troubleshooting tips

Illegal conversion occurs on any VARCHAR FOR BIT DATA column in a container-managed persistent bean

When enterprise beans with container-managed persistent (CMP) types that have any VARCHAR FOR BIT DATA columns defined on a DB2 table are deployed in the DB2 universal JDBC type 4 driver to persist the data, an SQLException of illegal conversion is thrown at run time. This exception only occurs when you use the DB2 universal JDBC type 4 driver and with the deferPrepares property being set to true. When the deferPrepares property is set to true, the DB2 universal JDBC type 4 driver uses the standard JDBC data mapping.

Currently, the generated deployed code does not follow the standard JDBC specification mapping. The failure at execution time is because of a problem in the tool that prepared the enterprise beans for execution.

To avoid receiving this exception, choose one of the following options:

- Set the deferPrepares property to false in the data source configuration.
- Do not use the DB2 universal JDBC type 4 driver if your table has any VARCHAR FOR BIT DATA or LONG VARCHAR FOR BIT DATA columns. Use the DB2 legacy CLI-based JDBC driver to persist the data. Refer to DB2 V8.1 readme for more details.

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 problems. Before opening a PMR, see the IBM Support page.

Cannot access a data source

What kind of database are you trying to access?

- Oracle
- DB2
- SQL Server
- Cloudscape
- Sybase
- General data access problems

If the errors described in the previous articles do not match the errors you see:

1. Browse the log files of the application server that contains the application, for clues. By default these files are *install_root*/*server_name*/SystemErr.log and SystemOut.log.
2. Browse the Helper Class property of the data source to verify that it is correct and that it is on the WebSphere Application Server class path. Mysterious errors or behavior might result from a missing or misnamed Helper Class name. If WebSphere Application Server cannot load the specified class, it uses a default helper class that might not function correctly with your database manager.
3. Verify that the Java Naming and Directory Interface (JNDI) name of the data source matches the name used by the client attempting to access it. If error messages indicate that the problem might be naming-related, such as referring to the **name server** or **naming service**, or including error IDs beginning with **NMSV**, look at the Naming related problems and Troubleshooting the naming service component topics.
4. Enable tracing for the resource adapter using the trace specification, RRA=all=enabled. Follow the instructions for dumping and browsing the trace output, to narrow the origin of the problem.

If none of these steps fixes your problem, see if the problem is identified and documented in available online support (hints and tips, technotes, and fixes). If none of the online resources listed in the topic describes your problem, see "Obtaining help from IBM" on page 155.

General data access problems
- An exception ″IllegalConnectionUseException″ occurs
- WTRN0062E: An illegal attempt to enlist multiple one phase capable resources has occurred.
- ConnectionWaitTimeoutException.
- com.ibm.websphere.ce.cm.StaleConnectionException: [IBM][CLI Driver] SQL1013N The database alias name or database name ″NULL″ could not be found. SQLSTATE=42705
- java.sql.SQLException: java.lang.UnsatisfiedLinkError:
- ″J2CA0030E: Method enlist caught java.lang.IllegalStateException″ wrapped in error ″WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable resources has occurred″ when attempting to execute a transaction.
- java.lang.UnsatisfiedLinkError:xaConnect exception when attempting a database operation
- ″J2CA0114W: No container-managed authentication alias found for connection factory or datasource *datasource*″ when attempting a database operation
- An error is thrown if you use the ws_ant command to perform the database customization for Structured Query Language in Java on HP platforms

IllegalConnectionUseException

This error can occur because a connection obtained from a WAS40DataSource is being used on more than one thread. This usage violates the J2EE 1.3 programming model, and an exception generates when it is detected on the server. This problem occurs for users accessing a data source through servlets or bean-managed persistence (BMP) enterprise beans.

To confirm this problem, examine the code for connection sharing. Code can inadvertently cause sharing by not following the programming model recommendations, for example by storing a connection in an instance variable in a servlet, which can cause use of the connection on multiple threads at the same time.

WTRN0062E: An illegal attempt to enlist multiple one phase capable resources has occurred

This error can occur because:
- An attempt was made to share a single-phase connection, when each **getConnection** method has different connection properties; such as the AccessIntent. This attempt causes a non-shareable connection to be created.
- An attempt was made to have more than one unshareable connection participate in a global transaction, when the data source is not an XA resource.
- An attempt was made to have a one-phase resource participate in a global transaction while an XA resource or another one-phase resource already participated in this global transaction.

  - Within the scope of a global transaction you try to get a connection more than once and at least one of the resource-refs you use specifies that the connection is unshareable, and the data source is not configured to support two-phase commit transactions. It does not support an XAResource. If you do not use a resource-ref, you default to unshareable connections.
  - Within the scope of a global transaction you try to get a connection more than once and at least one of the resource-refs you use specifies that the connection is shareable and the data source is not configured to support two-phase commit transactions. That is, it does not support an XAResource. In addition, even though you specify that connections are shareable, each getConnection request is made with different connection properties (such as IsolationLevel or AccessIntent). In this case, the connections are not shareable, and multiple connections are handed back.
  - Multiple components (servlets, session beans, BMP entity beans, or CMP entity beans) are accessed within a global transaction. All use the same data source, all specify shareable connections on their resource-refs, and you expect them to all share the same connection. If the properties are different, you get multiple connections. AccessIntent settings on CMP beans change their properties. To share a connection, the AccessIntent setting must be the same. For more information about CMP beans sharing a connection with non-CMP components, see the *Data access application programming interface support* and *Example: Accessing data using IBM extended APIs to share connections between container-managed and bean-managed persistence beans* topics in the DataAccess section of the information center.

To correct this error:
- Check what your client code passes in with its getConnection requests, to ensure they are consistent with each other.
- Check the connection sharing scope from the resource binding, using the .
  - If you are running an unshareable connection scope, verify that your data source is an XA data source.
  - If you are running a shareable connection scope, verify that all connection properties, including AccessIntent, are sharable.
- Check the JDBC provider implementation class from the Manage JDBC resource panel of the administrative console to ensure that it is a class that supports XA-type transactions.

ConnectionWaitTimeoutException accessing a data source or resource adapter

If your application receives exceptions like a com.ibm.websphere.ce.cm.ConnectionWaitTimeoutException or com.ibm.websphere.ce.j2c.ConnectionWaitTimeoutException when attempting to access a WebSphere Application Server data source or JCA-compliant resource adapter, respectively, some possible causes are:
- The maximum number of connections for a given pool is set too low. The demand for concurrent use of connections is greater then the configured maximum value for the connection pool. One indication that this situation is the problem is that you receive these exceptions regularly, but your CPU utilization is not high. This exception indicates that there are too few connections available to keep the threads in the server busy.
- Connection Wait Time is set too low. Current demand for connections is high enough such that sometimes there is not an available connection for short periods of time. If your connection wait timeout value is too low, you might timeout shortly before a user returns a connection back to the pool. Adjusting the connection wait time can give you some relief. One indication of this problem is that you use close to the maximum number of connections for an extended period and receiving this error regularly.
- You are not closing some connections or you are returning connections back to the pool at a very slow rate. This situation can happen when using unshareable connections, when you forget to close them, or you close them long after you are finished using them, keeping the connection from returning to the pool for reuse. The pool soon becomes empty and all applications get ConnectionWaitTimeoutExceptions. One indication of this problem is you run out of connections in the connection pool and you receive this error on most requests.

- You are driving more load than the server or backend system have resources to handle. In this case you must determine which resources you need more of and upgrade configurations or hardware to address the need. One indication of this problem is that the application or database server CPU is nearly 100% busy.

To correct these problems, either:
- Modify an application to use fewer connections
- Properly close the connections.
- Change the pool settings of MaxConnections or ConnnectionWaitTimeout.
- Adjust resources and their configurations.

com.ibm.websphere.ce.cm.StaleConnectionException: [IBM][CLI Driver] SQL1013N The database alias name or database name "NULL" could not be found. SQLSTATE=42705

This error occurs when a data source is defined but the **databaseName** attribute and the corresponding value are not added to the custom properties panel.

To add the **databaseName** property:
1. Click **Resources>Manage JDBC Providers** link in the administrative console.
2. Select the JDBC provider that supports the problem data source.
3. Select **Data Sources** and then select the problem data source.
4. Under **Additional properties** click **Custom Properties**.
5. Select the **databaseName** property, or add one if it does not exist, and enter the actual database name as the value.
6. Click **Apply** or **OK**, and then click **Save** from the action bar.
7. Access the data source again.

java.sql.SQLException: java.lang.UnsatisfiedLinkError:

This error indicates that the directory containing the binary libraries which support a database are not included in the LIBPATH environment variable for the environment in which the WebSphere Application Server starts.

The path containing the DBM vendor libraries vary by dbm. One way to find them is by scanning for the missing library specified in the error message. Then you can correct the LIBPATH variable to include the missing directory, either in the `.profile` of the account from which WebSphere Application Server is executed, or by adding a statement in a `.sh` file which then executes the `startServer` program.

"J2CA0030E: Method enlist caught java.lang.IllegalStateException" wrapped in error "WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable resources has occurred" when attempting to execute a transaction.

This error can occur when last participant support (LPS) is missing or disabled. LPS allows a one-phase capable resource and a two-phase capable resource to enlist within the same transaction.

LPS is only available if the following are true:
- WebSphere Application Server Programming Model Extensions (PME) is installed. PME is included in the Application Server Integration Server product.
- The Additional Integration Server Extensions option is enabled when PME is installed. If you perform a typical installation, this function is enabled by default. If you perform a custom installation, you have the option to disable this function, which disables LPS.
- The application enlisting the one-phase resource is deployed with the **Accept heuristic hazard** option enabled. This deployment is done through the Assembly Service Toolkit. To enable this option in the Assembly Service Toolkit:
  1. Load the `EAR` file into the Assembly Service Toolkit.

2. If the EAR file is actually a JTEE1.2 EAR then it must be upgraded to a JTEE1.3 EAR by clicking **File> Convert EAR** from the .
3. Select the EAR file in the left-hand panel of the Assembly Service Toolkit.
4. Select the **WAS Integration Server** tab in the bottom right window panel of the Assembly Service Toolkit
5. Ensure that the **Accept heuristic hazard** option is selected.
6. Save the EAR file.

java.lang.UnsatisfiedLinkError:xaConnect exception when attempting a database operation

This problem has two main causes:

- The most common cause is that the jdbc driver which supports connectivity to the database is missing, or is not the correct version, or that native libraries which support the driver are on the system's path.
  - To resolve this problem on a Windows platform, verify that the JDBC driver jar file is on the system PATH environment variable:
    - If you are using DB2,verify that at least the DB2 client product has been installed on the WebSphere host
      - On DB2 version 7.2 or earlier, the file where the client product is installed on the WebSphere Application Server is db2java.zip. Verify that the usejdbc2.bat program has been executed after the database install and after any upgrade to the database product.
      - On DB2 version 8.1 or later, use the DB2 Universal JDBC Provider Driver when defining a JDBC provider in WebSphere Application Server. The driver file is db2jcc.jar. If you use the type 2 (default) option, verify that at least the DB2 client product is installed on the WebSphere Application Server host. If you specify the type 4 option, the DB2 client does not need to be installed, but the file db2jcc.jar still must be present.

        When specifying the location of the driver file, it is recommended to that you specify the path and file name of the target DB2 installation, rather than simply copying the file to a local directory, if possible. Otherwise, you may be exposed to problems if the target DB2 installation is upgraded and the driver used by WebSphere Application Server is not. If you choose **DB2 Legacy CLI-based type 2 JDBC Driver** when defining a JDBC provider in WebSphere Application Server, then you must still follow the steps to ensure that you have the correct version of the db2java.zip file (see instructions for DB2 7.2 or earlier).
  - On a Unix platform, ensure that any native libraries required to support the database client of your database product are specified in the LD_LIBRARY_PATH environment variable in the profile of the account under which WebSphere Application Server executes.

    If you are using DB2 The native library is libdb2jdbc.so. The best way to ensure that this library is accessed correctly by WebSphere is to call the db2profile script supplied with DB2 from the .profile script of the account (such as ″root″) under which WebSphere runs.
    - If you are using DB2 version 7.2 or earlier, ensure that the usejdbc2,script provided with DB2 is called from the profile of the account under which WebSphere Application server is launched.
    - If you are using DB2 version 8.1 or later, see the previous instructions for the Windows operating system.
- If the database manager is DB2, you may have chosen the option to create a 64-bit instance. A 64-bit configuration is not supported. If this has happened, remove the database instance and create a new one with the default 32-bit setting.

Note: For general help in configuring JDBC drivers and data sources in WebSphere Application Server, see the topic Accessing data from applications.

"J2CA0114W: No container-managed authentication alias found for connection factory or datasource *datasource*" when attempting a database operation

This error might occur in the SystemOut.log file when you run an application to access a data source after creating the data source using JACL script.

The error message occurs because the JACL script did not set container-managed authentication alias for CMP connection factory. The JACL is missing the following line:

```
$AdminConfig create MappingModule $cmpConnectorFactory  "{mappingConfigAlias
DefaultPrincipalMapping}{authDataAlias $authDataAlias}
```

To correct this problem, add the missing line to the JACL script and run the script again. See Example: Creating a JDBC provider and data source using Java Management Extensions API and the scripting tool for a sample JACL script.

An error is thrown if you use the ws_ant command to perform the database customization for Structured Query Language in Java on HP platforms

If you use the `ws_ant` command to perform the database customization for Structured Query Language in Java (SQLJ) on HP platforms, you can receive an error similar to the following:

```
[java] [ibm][db2][jcc][sqlj]
[java] [ibm][db2][jcc][sqlj] Begin Customization
[java] [ibm][db2][jcc][sqlj] encoding not supported!!
```

The cause of this error might be that your databases were created using the HP default character set. The Java Common Client (JCC) driver depends on the software development kit (SDK) to perform the codepage conversions. The SDK shipped with this product, however, does not support the HP default codepage.

You need to set your LANG to the ISO locale before creating the databases. It should be similar to the following:

```
export LANG=en_US.iso88591
```

Refer to the DB2 Tech Notes at `http://www-3.ibm.com/software/data/db2/udb/ad/v8/bldg/t0004877.htm` for details.

Problems accessing an Oracle data source

What kind of error do you see when you try to access your Oracle-based data source

- An Invalid Oracle URL is specified
- ″DSRA0080E: An exception was received by the Data Store Adapter. See original exception message: ORA-00600″ when connecting to or using an Oracle data source.
- ″DSRA8100E: Unable to get a {0} from the DataSource. Explanation: See the linkedException for more information.″
- An ″Error while trying to retrieve text for error″ message occurs when connecting to an Oracle data source.
- A java.lang.UnsatisfiedLinkError occurs when connecting to an Oracle data source.
- The exception java.lang.NullPointerException or an ″internal error: oracle.jdbc.oci8.OCIEnv″ occurs when connecting to an Oracle data source.
- WSVR0016W: Classpath entry, ${ORACLE_JDBC_DRIVER_PATH}/classes12.zip, in Resource, Oracle JDBC Thin Driver, located at cells/BaseApplicationServerCell/nodes/wasrtp/resources.xml has an invalid variable.

An invalid Oracle URL is specified

This error might be caused by an incorrectly specified URL on the URL property of the target data source.

Examine the URL property for the data source object in the administrative console. For the 8i OCI driver, verify that **oci8** is used in the URL. For the 9i OCI driver, you can use either **oci8** or **oci**.

Examples of Oracle URLs:
- For the thin driver: `jdbc:oracle:thin:@hostname.rchland.ibm.com:1521:IBM`
- For the thick (OCI) driver: `jdbc:oracle:oci8:@tnsname1`

"DSRA0080E: An exception was received by the data store adapter. See original exception message: ORA-00600" when connecting to or using an Oracle data source "DSRA0080E: An exception was received by the data store adapter. See original exception message: ORA-00600" when connecting to or using an Oracle data source "DSRA0080E: An exception was received by the Data Store Adapter. See original exception message: ORA-00600" when connecting to or using an Oracle data source

A possible reason for this exception is that the version of the Oracle JDBC driver being used is older than the Oracle database. It is possible that more than one version of the Oracle JDBC driver is configured on the WebSphere Application Server.

Examine the version of the JDBC driver. Sometimes you can determine the version by looking at the class path to determine what directory the driver is in.

If you cannot determine the version this way, use the following program to determine the version. Before running the program, set the class path to the location of your JDBC driver files.

```
import java.sql.*;
     import oracle.jdbc.driver.*;
     class JDBCVersion
     {
         public static void main (String args[])
         throws SQLException
         {
             // Load the Oracle JDBC driver
             DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
             // Get a connection to a database
             Connection conn = DriverManager.getConnection
             ("jdbc:oracle:thin:@appaloosa:1521:app1","sys","change_on_install");
             // Create Oracle DatabaseMetaData object
             DatabaseMetaData meta = conn.getMetaData();
             // gets driver info:
             System.out.println("JDBC driver version is " + meta.getDriverVersion());
         }
     }
```

If the driver and the database are at different versions, replace the JDBC driver with the correct version. If multiple drivers are configured, remove any that occur at the incorrect level.

DSRA8100E: Unable to get a {0} from the DataSource. Explanation: See the linkedException for more information.

When using an oracle thin driver, Oracle throws a ″java.sql.SQLException: invalid arguments in call″ error if no user name or password is specified when getting a connection. If you see this error while running WebSphere Application Server, the alias is not set.

To remove the exception, define the alias on the data source.

"Error while trying to retrieve text for error" error when connecting to an Oracle data source

The most likely cause of this error is that the Oracle 8i OCI driver is being used with an ORACLE_HOME property that is either not set or is set incorrectly.

To correct the error, examine the user profile that WebSphere Application Server is running under to verify that the $ORACLE_HOME environment variable is set correctly.

"java.lang.UnsatisfiedLinkError:" connecting to an Oracle data source

The environment variable LIBPATH might not be set or is set incorrectly, if your data source throws an **UnsatisfiedLinkError** error, and the full exception indicates that the problem is related to an Oracle module, as in the following examples.

Example of invalid an LIBPATH for the 8i driver:

```
Exception in thread "main" java.lang.UnsatisfiedLinkError:
/usr/WebSphere/AppServer/java/jre/bin/libocijdbc8.so:
load ENOENT on shared library(s)
/usr/WebSphere/AppServer/java/jre/bin/libocijdbc8.so libclntsh.a
```

Example of an invalid LIBPATH for the 9i driver:

```
Exception in thread "main" java.lang.UnsatisfiedLinkError:
no ocijdbc9  (libocijdbc9.a or .so) in java.library.path
at java.lang.ClassLoader.loadLibrary(ClassLoader.java(Compiled Code))
at java.lang.Runtime.loadLibrary0(Runtime.java:780)
```

To correct the problem, examine the user profile under which the WebSphere Application Server is running to verify that the LIBPATH environment variable includes Oracle libraries. Scan for the `lobocijdbc8.so` file to find the right directory.

java.lang.NullPointerException referencing 8i classes, or " internal error: oracle.jdbc.oci8. OCIEnv" connecting to an Oracle data source

The problem might be that the 9i OCI driver is being used on an AIX 32-bit machine, the LIBPATH is set correctly, but the ORACLE_HOME environment variable is not set or is set incorrectly. You can encounter an exception similar to either of the following when your application attempts to connect to an Oracle data source:

Exception example for the java.lang.NullPointerException:

```
Exception in thread "main" java.lang.NullPointerException
at oracle.jdbc.oci8.OCIDBAccess.check_error(OCIDBAccess.java:1743)
at oracle.jdbc.oci8.OCIEnv.getEnvHandle(OCIEnv.java:69)
at oracle.jdbc.oci8.OCIDBAccess.logon(OCIDBAccess.java:452)
at oracle.jdbc.driver.OracleConnection. <init>(OracleConnection.java:287)
```

Exception example for the java.sql.SQLException:

```
Exception in thread "main" java.sql.SQLException:
internal error: oracle.jdbc.oci8. OCIEnv@568b1d21
at oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:184)
at oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:226)
at oracle.jdbc.oci8.OCIEnv.getEnvHandle(OCIEnv.java:79)
```

To correct the problem, examine the user profile that WebSphere Application Server is running under to verify that it has the `$ORACLE_HOME` environment variable set correctly, and that the `$LIBPATH` includes `$ORACLE_HOME/lib`.

WSVR0016W: Classpath entry, ${ORACLE_JDBC_DRIVER_PATH}/classes12.zip, in Resource, Oracle JDBC Thin Driver, located at cells/BaseApplicationServerCell/nodes/wasrtp/resources.xml has an invalid variable

This error occurs when there is no environment variable defined for the property, ORACLE_JDBC_DRIVER_PATH.

Verify this problem in the administrative console. Go to **Environment > Manage WebSphere Variables** to verify whether the variable `ORACLE_JDBC_DRIVER_PATH` is defined.

To correct the problem, click **New** and define the variable. For example, name : **ORACLE_JDBC_DRIVER_PATH** , value : **c:\oracle\jdbc\lib** Use a value that names the directory in your operating system and directory structure that contains the `classes12.zip` file.

Problems accessing a DB2 database

What kind of problem are you having accessing your DB2 database?

- SQL0567N ″DB2ADMIN″ is not a valid authorization ID. SQLSTATE=42602.
- SQL0805N Package ″*package name*″ was not found.
- SQL0805N Package ″NULLID.SQLLC300″ was not found. SQLSTATE=51002.
- SQL30082N Attempt to establish connection failed with security reason ″17″ (″UNSUPPORTED FUNCTION’) SQLSTATE=08001
- SQLException, with ErrorCode -99,999 and SQLState 58004, with Java ″StaleConnectionException: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=58004″ using WAS40-type data source.
- Error message `java.lang.reflect.InvocationTargetException: com.ibm.ws.exception.WsException: DSRA0023E: The DataSource implementation class ″COM.ibm.db2.jdbc.DB2XADataSource″ could not be found.` when trying to access a DB2 database
- CLI0119E System error. SQLSTATE=58004 - DSRA8100 : Unable to get a XAconnection, or DSRA0011E: Exception: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=58004.
- COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver][DB2/NT] SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code ″2″. SQLSTATE=40001.
- ″COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource″ could not be found for data source (*data_source*).
-
- java.sql.SQLException: Failure in loading T2 native library db2jcct2 DSRA0010E: SQL State = null, Error Code = -99,999
- Lock contention exception occurs in database when data source implementation type is XA
- ″DSRA8050W: Unable to find the DataStoreHelper class specified″ exception occurs when trying to use a DB2 Universal Datasource in a mixed release cell.
- Check the IBM support page for information on problems that occur when using connection pooling with DB2.

SQL0567N "DB2ADMIN " is not a valid authorization ID. SQLSTATE=42602

If you encounter this error when attempting to access a DB2 Universal Database (UDB):

1. Verify that your user name and password in the data source properties page in the administrative console are correct.
2. Ensure that the user ID and password do not contain blank characters before, in between, or after.

SQL0805N Package *package-name* was not found

Possible reasons for these exceptions:

- If the package name is `NULLID.SQLLC300`, see SQL0805N Package ″NULLID.SQLLC300″ was not found. SQLSTATE=51002. for the reason.
- You are attempting to use an XA-enabled JDBC driver on a DB2 database that is not XA-ready.

To correct the problem on a DB2 Universal Database (UDB), run this one-time procedure, using the db2cmd interface while connected to the database in question:

1. **DB2 bind @db2ubind.lst blocking all grant public**
2. **DB2 bind @db2cli.lst blocking all grant public**

The `db2ubind.lst` and `db2cli.lst` files are in the `bnd` directory of your DB2 installation root. Run the commands from that directory.

SQL0805N Package "NULLID.SQLLC300" was not found. SQLSTATE=51002

This error can occur because:
- The underlying database was dropped and recreated.
- DB2 was ugpraded and its packages are not rebound correctly.

To resolve this problem, rebind the DB2 packages by running the db2cli.lst script found in the bnd directory. For example: db2>@db2cli.lst.

SQL30082N Attempt to establish connection failed with security reason "17" ("UNSUPPORTED FUNCTION') SQLSTATE=08001

This error can occur when the security mechanism specified by the client is not valid for this server. Some typical examples:
- The client sent a new password value to a server that does not support the change password function.
- The client sent SERVER_ENCRYPT authentication information to a server that does not support password encryption.
- The client sent a userid, but no password, to a server that does not support authentication by userid only.
- The client has not specified an authentication type, and the server has not responded with a supported type. This can include the server returning multiple types from which the client is unable to choose.

To resolve this problem, ensure that your client and server use the same security mechanism. For example, if this is an error on your data source, verify that you have assigned a user id and password or authentication alias.

SQLException, with ErrorCode -99,999 and SQLState 58004, with Java "StaleConnectionException: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=58004", when using WAS40-type data source

An unexpected system failure usually occurs when running in XA mode (two-phase commit). Among the many possible causes are:
- An invalid username or password was provided.
- The database name is incorrect.
- Some DB2 packages are corrupted.

To determine whether you have a user name or password problem, look in the `db2diag.log` file to view the actual error message and SQL code. A message like the following example, with an SQLCODE of -1403, indicates an invalid user ID or password:

```
2002-07-26-14.19.32.762905   Instance:db2inst1   Node:000
PID:9086(java)   Appid:*LOCAL.db2inst1.020726191932
XA DTP Support  sqlxa_open   Probe:101
DIA4701E Database "POLICY2" could not be opened
for distributed transaction processing.
String Title: XA Interface SQLCA  PID:9086 Node:000
SQLCODE = -1403
```

To resolve these problems:
1. Correct your user name and password. If you specify your password on the GUI for the data source, ensure that the user name and password you specify on the bean are correct. The user name and password you specify on the bean overwrite whatever you specify when creating the data source.
2. Use the correct database name.
3. Rebind the packages (in the `bnd` directory) as follows:

```
db2connect to dbname
c:\SQLLIB\bnd>DB2 bind @db2ubind.lst blocking all grant public
c:\SQLLIB\bnd>DB2 bind @db2cli.lst blocking all grant public
```

4. Ensure that the \WebSphere\AppServer\properties\wsj2cdpm.properties file has the right user ID and password.

Error message `java.lang.reflect.InvocationTargetException: com.ibm.ws.exception.WsException: DSRA0023E: The DataSource implementation class "COM.ibm.db2.jdbc.DB2XADataSource" could not be found.` when trying to access a DB2 database

One possible reason for this exception is that a user is attempting to use a JDBC 2.0 DataSource, but DB2 is not JDBC 2.0-enabled. This situation frequently happens with new installations of DB2 because DB2 provides separate drivers for JDBC 1.X and 2.0, with the same physical file name. By default, the JDBC 1.X driver is on the class path.

To confirm this problem:
- On Windows systems, look for the inuse file in the java12 directory in your DB2 installation root. If the file missing, you are using the JDBC 1.x driver.
- On UNIX systems, check the class path for your data source. If the class path does not point to the `db2java.zip` file in the `java12` directory, you are using the JDBC 1.x driver.

To correct this problem:
- On Windows systems, stop DB2. Run the usejdbc2.bat file from the java12 directory in your DB2 instalation root. Run this file from a command line to verify that it completes successfully.
- On UNIX systems, change the class path for your data source to point to the `db2java.zip` file in the `java12` directory of your DB2 installation root.

CLI0119E System error. SQLSTATE=58004 - DSRA8100 : Unable to get a XAconnection or DSRA0011E: Exception: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=5800

If you encounter this error when attempting to access a DB2 Universal Database (UDB) data source:
1. Check your user name and password custom properties in the data source properties page in the administrative console. Verify that they are correct.
2. Ensure the user ID and password do not contain any blank characters, before, in between, or after.
3. Check that the `WAS.policy` file exists for the application, for example, `D:\WebSphere\AppServer\installedApps\markSection.ear\META-INF\was.policy`.
4. View the entire exception listing for an underlying SQL error, and look it up using the DBM vendor message reference.

If you encounter this error while running DB2 on Red Hat Linux, the **max queues system wide** parameter is too low to support DB2 while it acquires the necessary resources to complete the transaction. When this problem exists, the exceptions J2CA0046E and DSRA0010E can precede the exception DSRA8100E.

To correct this problem, edit the /proc/sys/kernal/msgmni file to increase the value of the **max queues system wide** parameter to a value greater than 128.

COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver][DB2/NT] SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code "2". SQLSTATE=40001

This problem is probably an application-caused DB2 deadlock, particularly if you see an error similar to the following when accessing a DB2 data source:

```
ERROR CODE: -911
COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver][DB2/NT] SQL0911N
The current transaction has been rolled back because of a deadlock or timeout.
Reason code "2".  SQLSTATE=40001
```

To diagnose the problem:
1. Execute these DB2 commands:

a. `db2 update monitor switches using LOCK ON`
b. `db2 get snapshot for LOCKS on dbName >`

The *directory_name*`\lock_snapshot.log` now has the DB2 lock information.

2. Turn off the lock monitor by executing: `db2 update monitor switches using LOCK OFF`

To verify that you have a deadlock:

1. Look for an application handle that has a lock-wait status, and then look for the ID of the agent holding lock to verify the ID of the agent.
2. Go to that handle to verify it has a lock-wait status, and the ID of the agent holding the lock for it. If it is the same agent ID as the previous one, then you know that you have a circular lock (deadlock).

To resolve the problem:

1. Examine your application and use a less restrictive isolation level if no concurrency access is needed.
2. Use caution when changing the **accessIntent** value to move to a lower isolation level. This change can result in data integrity problems.
3. For DB2/UDB Version 7.2 and earlier releases, you can set the DB2_RR_TO_RS flag from the DB2 command line window to eliminate unnecessary deadlocks, such as when the accessIntent defined on the bean method is too restrictive, for example, PessimisticUpdate. The DB@_RR_TO_RS setting has two impacts:
   - If RR is your chosen isolation level, it is effectively downgraded to RS.
   - If you choose another isolation level, and the DB2_RR_TO_RS setting is on, a scan skips over rows that are deleted but not committed, even though the row might qualify for the scan. The skipping behavior affects the RR, Read Stability (RS), and Cursor Stability (CS) isolation levels.

   For example, consider the scenario where transaction A deletes the row with column1=10 and transaction B does a scan where column1>8 and column1<12. With DB2_RR_TO_RS off, transaction B waits for transaction A to commit or rollback. If transaction A rolls back, the row with column1=10 is included in the result set of the transaction B query. With DB2_RR_TO_RS on, transaction B does not wait for transaction A to commit or rollback. Transaction B immediately receives query results that do not include the deleted row. Setting DB2_RR_TO_RS effectively changes locking behavior, thus avoiding deadlocks.

"COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource" could not be found for data source ([data-source-name])"

This error is denoted by message DSRA8040I: Failed to connect to the DataSource.

This error usually occurs when the class path of the DB2 JDBC driver is set correctly to `${DB2_JDBC_DRIVER_PATH}/db2java.zip` but the environment variable DB2_JDBC_DRIVER_PATH is not set.

This error can also occur if you are using DB2 Version 7.1 or 7.2 and you have not yet run *usejdbc2*. This might be the problem if your path is correct but you still receive this error.

To confirm this problem:

1. Go to the **Manage WebSphere Variables** panel.
2. Select **Environment** to verify that there is no entry for the variable DB2_JDBC_DRIVER_PATH.

To correct this problem: Add the variable DB2_JDBC_DRIVER_PATH with **value** equal to the directory path containing the `db2java.zip` file.

If the path is correct and you need to run usejdbc2, refer to Configuring WebSphere Application Server for DB2 access.

java.sql.SQLException: Failure in loading T2 native library db2jcct2 DSRA0010E: SQL State = null, Error Code = -99,999

The *Failure in loading* message indicates one of two things:

- Usually this happens when the machine was not rebooted after installing DB2. Reboot the machine getting the error and try it again.
- The DB2 context is not getting set up correctly for the user running WebSphere Application Server. Source the db2profile file on the machine, and ensure that the environment contains pointers to the DB2 native libraries.

Lock contention exception occurs in database when data source implementation type is XA

Note: Because a lock contention exception can be caused by many factors, consider the following explanation and recommended response as a strategy for eliminating the possible reasons for your lock contention problem.

Symptom	A lock contention exception occurs in a DB2 database that your application accesses through a data source of implementation type XA.
Problem	Your application is trying to access database records that are locked by an XA transaction that is in ended (e) state, but cannot be prepared by the transaction manager.
Description	An XA transaction to DB2 that ends, but cannot be prepared, is in ended (e) state. Because it is *not* considered to be *in doubt*, the transaction manager cannot recover this transaction. DB2 does not return it in the list of in doubt transactions. DB2 also does not roll the transaction back immediately; it waits until all connections to the database are released. During this period of inaction, the transaction continues to hold locks on the database. Due to certain policies of WebSphere Application Server workload management, your application server might not disconnect all connections from the database to allow rollback of the transaction. Therefore the ended transaction persists in locking the same database records. If your application attempts to access these locked records, a lock contention exception occurs in DB2.
Recommended response	DB2 Version 8.2 is shipped with a sample application that connects to a defined DB2 server and uses the available DB2 APIs to obtain a list of these particular ended transactions. The application offers a configuration setting that enables you to designate an amount of time after which the application rolls these transactions back. Locate the sample application in the `sqllib/samples/db2xamon.c` directory of DB2 Version 8.2 and run it.

"DSRA8050W: Unable to find the DataStoreHelper class specified" exception occurs when trying to use a DB2 Universal Datasource in a mixed release cell.

This error usually occurs when you are using WebSphere Application Server Version 6 in conjunction with a previous version and attempt to create a DB2 Universal Datasource on the previous version.

This can happen because the DB2 Universal Datasource was not available on Version 5 and previous versions, but the Version 6 administrative console allows you to build one.

To correct this problem: create the datasource on Version 6.

Problems accessing a SQL server data source

What kind of problem are you having accessing your SQL Server database?

- ERROR CODE: 20001 and SQL STATE: HY000.
- Application fails with message stating ″Cannot find stored procedure...″
- ERROR CODE: SQL5042 when running a Java application

ERROR CODE: 20001 and SQL STATE: HY000 accessing SQLServer database

The problem might be that the distributed transaction coordinator service is not started. Look for an error similar to the following example when attempting to access an SQL server database:

```
ERROR CODE: 20001
SQL STATE:  HY000
java.sql.SQLException: [Microsoft][SQLServer JDBC Driver]
[SQLServer]xa_open (0) returns -3
at com.microsoft.jdbc.base.BaseExceptions.createException(Unknown Source) ...
at com.microsoft.jdbcx.sqlserver.SQLServerDataSource.getXAConnection
(Unknown Source) ...
```

To confirm this problem:

1. Go to the Windows **Control Panel** and click **Services**(or click **Control Panel > Administrative Tools > Services**)
2. Verify whether the service **Distributed Transaction Coordinator** or **DTC** is started.
3. If not, start the Distributed Transaction Coordinator service.

Application fails with message stating "Cannot find stored procedure..." accessing an SQLServer database

This error can occur because the stored procedures for the Java Transaction API (JTA) feature are not installed on the Microsoft SQL Server.

To resolve the problem: Repeat the installation for the stored procedures for the JTA feature, according to the ConnectJDBC installation guide.

ERROR CODE: SQL5042 when running a Java application

This error can occur when you configure your application to run in the following manner:

1. you use a type 2 (application) driver running on the gateway to the OS 390
2. your application is an XA application.

OS 390 does not use XA, but uses SPM. To resolve the problem:

1. Check your dbm cfg to see that the SPM is not started on the gateway.
2. Assign a port and set the *db2comm* variable to **TCPIP**.
3. Update the dbm cfg value *SPM_NAME* to use your machine name.
4. Start the SPM on the gateway.

Problems accessing a Cloudscape database

What kind of problem are you having accessing your Cloudscape database?

- Unexpected IOException wrapped in SQLException, accessing Cloudscape database.
- The ″Select for update″ operatioin on one row causes table to become locked, triggering a deadlock condition.
- ″ERROR XSDB6: Another instance of Cloudscape might have already booted the database *databaseName*.″ error starting application server.

- Error ″The version of the IBM Universal JDBC driver in use is not licensed for connectivity to Cloudscape databases″
- Running an application causes a runtime exception which produces an unreadable message.

Tip: Cloudscape errorCodes (2000, 3000, 4000) indicate levels of severity, not specific error conditions. In diagnosing Cloudscape problems, pay attention to the given sqlState value.

Unexpected IOException wrapped in SQLException, accessing Cloudscape database

This problem can occur because Cloudscape databases use a large number of files. Some operating systems, such as the Solaris Operating Environment, limit the number of files an application can open at one time. If the default is a low number, such as 64, you can get this exception.

If your operating system lets you configure the number of file descriptors, you can correct the problem by setting the number to a high value, such as 1024.

The "select for update" operation causes table lock and deadlock when accessing Cloudscape

If a select for update operation on one row locks the entire table, which creates a deadlock condition, there might be undefined indexes on that table. The lack of an index on the columns you use in the `where` clause can cause Cloudscape to create a table lock rather than a row level lock.

To resolve this problem, create an index on the affected table.

ERROR XSDB6: Another instance of Cloudscape may have already booted the database "database"

This problem occurs because Cloudscape embedded framework only allows one Java virtual machine (JVM) to access the database instance at a time.

To resolve this problem:
1. Verify that you do not have other JDBC client programs, such as **ij** or **cview** running on that database instance, when WebSphere Application Server is running.
2. Verify that you do not use the same instance of the database for more than one data source or use the networkServer framework, which doesn't have this limitation.
3. If there are no connections to Cloudscape, delete the db.lck lock file. This file can be found in the directory where the Cloudscape database is mounted, under the schema directory. For example, if the database is mounted at /myCloudscapeDB, issue the command: rm /myCloudscapeDB/schemaName/db.lck

Error ″The version of the IBM Universal JDBC driver in use is not licensed for connectivity to Cloudscape databases″

Error "The version of the IBM Universal JDBC driver in use is not licensed for connectivity to Cloudscape databases"

At the client runtime, an error similar to the following occurs:

```
The version of the IBM Universal JDBC driver in use is not
licensed for connectivity to Cloudscape databases.  To connect
to this DB2 server, please obtain a licensed copy of the IBM DB2
Universal Driver for JDBC and SQLJ.  An appropriate license file
db2jcc_license_*.jar for this target platform must be installed to
the application classpath.  Connectivity to Cloudscape databases is
enabled by any of the following license files:
{ db2jcc_license_c.jar, b2jcc_license_cu.jar, db2jcc_license_cisuz.jar }
```

The problem occurs because an incorrect JDBC driver jar file name is specified in the class path for JDBC provider. For example, the jar file name may have an extra '_', as follows:

```
${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license__cu.jar
```

To resolve the problem:

1. Correct the UNIVERSAL_JDBC_DRIVER_PATH jar file name in the JACL script
2. Restart the cluster.
3. Rerun the client.

Running an application causes a runtime exception which produces an unreadable message.

At client runtime, you may receive a message similar to the following: `Caused by: com.ibm.db2.jcc.a.SqlException: DB2 SQL error: SQLCODE: -1, SQLSTATE: 42X05, SQLERRMC: ANNUITYHOLDER20^T42X05`

The problem occurs because the property *retrieveMessagesfromServerOnGetMessage*, which is required by WebSphere Application Server, has not been set.

To resolve the problem, on the admininstrative console

1. Click **Resources -> JDBC Providers**
2. Click on a Cloudscape provider
3. Scroll down and click on **Data Sources**
4. Select your data source (or add a new one)
5. Scroll down and select **Custom Properties**
6. If the property *retrieveMessagesFromServerOnGetMessage* already exists, set its value to true. If the property does not exist, select **New**and add the property *retrieveMessagesFromServerOnGetMessage* with a value **true**
7. Rerun the client

The SystemOut.log will now generate readable messages so that you can resolve the underlying problem.

Problems accessing a Sybase data source

What kind of problem are you having accessing your Sybase database?

- ″Sybase Error 7713: Stored Procedure can only be executed in unchained transaction mode″ error.
- ″JZ0XS: The server does not support XA-style transactions. Please verify that the transaction feature is enabled and licensed on this server.″
- A container managed persistence (CMP) enterprise bean is causing exceptions.

"Sybase Error 7713: Stored Procedure can only be executed in unchained transaction mode" error

This error occurs when either:

- The JDBC attempts to put the connection in **autocommit(true)** mode.
- A stored procedure is not created in a compatible mode.

To fix the **autocommit(true)** mode problem, let the application change the connection to chained mode using the **Connection.setAutoCommit(false)** mode, or use a **set chained on** language command.

To resolve the stored procedure problem, use the `sp_procxmode` *`procedure_name`* `"anymode"` command.

"JZ0XS: The server does not support XA-style transactions. Please verify that the transaction feature is enabled and licensed on this server."

This error occurs when XA-style transactions are attempted on a server that does not have Distributed Transaction Management (DTM) installed.

To resolve this problem, use the instructions in the Sybase Manual titled: *Using Adaptive Server Distributed Transaction Management Features* to enable Distributed Transaction Management (DTM). The main steps in this procedure are:
1. Install the DTM option.
2. Check the `license.dat` file to verify that the DTM option is installed.
3. Restart the license manager.
4. Enable DTM in ISQL.
5. Restart the ASE service.

A container managed persistence (CMP) enterprise bean is causing exceptions

This error is caused by improper use of reserved words. Reserved words cannot be used as column names.

To correct this problem: Rename the variable to remove the reserved word. You can find a list of reserved words in the *Sybase Adaptive Server Enterprise Reference Manual; Volume 1: Building Blocks*, Chapter 4. This manual is available online at: http://manuals.sybase.com/onlinebooks/group-as/asg1250e/refman.

Sybase troubleshooting tips

This article describes how to diagnose the following problems related to Sybase:
- Executing the DatabaseMetaData.getBestRowIdentifier() method in an XA transaction causes errors
- Sybase requirements for using the escapes and DatabaseMetaData methods
- Database deadlocks and `XA_PROTO` errors occur when using Sybase
- Executing a stored procedure containing a **SELECT INTO** command causes exception
- Error is incorrectly reported about IMAGE to VARBINARY conversion
- JDBC 1.0 standard methods are not implemented and generate a SQL exception when used
- Sybase transaction manager fails after trying to alleviate deadlock error
- Starting an XA transaction when the autoCommit value of the connection is *false* causes error
- Sybase does not throw an exception when an incorrect database name is specified

Executing the DatabaseMetaData.getBestRowIdentifier() method in an XA transaction causes errors

Executing the DatabaseMetaData.getBestRowIdentifier() method while in an XA transaction causes the following errors:

```
SQL Exception: The 'CREATE TABLE' command is not allowed within a
multi-statement transaction in the 'tempdb' database.  Calling
DatabaseMetaData.getBestRowIdentifier()
```

Currently, this method fails when using Sybase. This problem occurs with other methods as well, including:
- getBestRowIdentifier();
- getVersionColumns();
- getTablePrivileges();
- getProcedureColumns();
- getPrimaryKeys();
- getIndexInfo();
- getImportedKeys();
- getExportedKeys();
- getCrossReference();
- getColumns();
- getColumnPrivileges();

Case 10880427 has been opened with Sybase to resolve this problem.

Sybase requirements for using the escapes and DatabaseMetaData methods

To use the escapes and DatabaseMetaData methods, you must install stored procedures on the Adaptive Server Enterprise or Adaptive Server Anywhere database where you want to use these methods. These stored procedures are also required by some of the connection methods.

To check for the presence of LOCATE ():
1. Open a Sybase **isql** command prompt.
2. Type the command **use master**.
3. Type the command **go**.
4. Type the SQL command and select * from jdbc_function_escapes.
5. Type the command **go**.

The following appears:

```
escape_name      map_string
 --------------------------------
 abs                    abs(%1)
 acos                  acos(%1)
 asin                   asin(%1)
 atan                   atan(%1)
 atan2                 atn2(%1, %2)
 ceiling                ceiling(%1)
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

locate charindex ((convert (varchar, %1)), (convert (varchar, %2)))
```

If the function does not exist, upgrade jConnect to at least Version 5.2 EBF 10635 and run the following command:

```
java IsqlApp -U sa -P -S jdbc:sybase:Tds:hostname:4100 -I %JDBC_HOME%\sp\sql_server12.sql -c go
```

Database deadlocks and XA_PROTO errors occur when using Sybase

When using Sybase with the IBM WebSphere Application Server, do one of the following to prevent database deadlocks and errors:
- Change the transaction isolation level on the connection to TRANSACTION_READ_COMMITTED. Set the isolation level on the connection for unshareable connections or, for shareable connections, define the isolation levels in the resource reference for your data source using an assembly tool.
- Modify Sybase by doing one of the following:
  - If you want to use the existing tables, modify the table locking scheme using the **alter table** *table name* **lock datarows** command to get a row lock level granularity.
  - If you want to set the system-wide locking scheme to data rows, all subsequently created tables inherit that value and have a locking scheme of data rows.

    Note: You must drop your original databases and tables.

Executing a stored procedure containing a SELECT INTO command causes exception

An attempt to execute a stored procedure containing a **SELECT INTO** command results in the following exception:

```
SVR-ERROR: SQL Exception SELECT INTO command not allowed within multi-statement transaction
```

Case 10868947 has been opened with Sybase to resolve this problem.

Error is incorrectly reported about IMAGE to VARBINARY conversion

The following error is incorrectly reported:

```
com.sybase.jdbc2.jdbc.SybSQLException: Implicit conversion from data type 'IMAGE' to 'VARBINARY' is not allowed.
 Use the CONVERT function to run this query.
```

The error is about a VARBINARY column only and causes confusion if you also have an IMAGE column.

Do one of the following to work around this problem:
- Use a PreparedStatement.setBytes() method instead of a PreparedStatement.setBinaryStream() method
- Use a LONG VARBINARY for the column type if you want to continue using the setBinaryStream() method. You might want to make this change because the size limit for VARBINARY is 255 bytes.

For example:

```
// ***************CORRECTION*****************************
        // setBinaryStream fails for column type of VARBINARY , use setBytes() instead
        //stmt4.setBinaryStream(8,new java.io.ByteArrayInputStream(tempbyteArray),tempbyteArray.length);
        stmt4.setBytes(8,tempbyteArray);
```

JDBC 1.0 standard methods are not implemented and generate a SQL exception when used

The following JDBC 1.0 standard methods are not implemented and generate a SQL exception when used:
- ResultSetMetaData.getSchemaName()
- ResultSetMetaData.getTableName() (implemented only for text and image datatypes)
- ResultSetMetaData.getCatalogName()

Sybase transaction manager fails after trying to alleviate a deadlock error

If an application encounters a deadlock, Sybase detects the deadlock and throws an exception. Because of this detection, the transaction manager calls an `xa_end` with a `TMFAIL` in it.

The call succeeds, but causes another Sybase exception, `XAERR_PROTO`. This exception only appears in the error log and does not cause any functional problems. All applications should continue to run, therefore no workaround is necessary.

Case 10869169 has been opened with Sybase to resolve this problem.

Starting an XA transaction when the autoCommit value of the connection is *false* causes error

The exception thrown is `javax.transaction.xa.XAException` with stack trace similar to the following:

```
at com.sybase.jdbc2.jdbc.SybXAResource.sendRPC(SybXAResource.java:711)
 at com.sybase.jdbc2.jdbc.SybXAResource.sendRPC(SybXAResource.java:602)
 at com.sybase.jdbc2.jdbc.SybXAResource.start(SybXAResource.java:312)
```

This problem affects you when you do both local and global transactions. If, in a local transaction, the autoCommit default value is set to *false*, and a global or XA transaction starts (either a user transaction started by you, or a container transaction started by a container), the exception occurs.

This problem is a Sybase bug as the start() method can fail unexpectedly, regardless of the value of autoCommit. Currently, there is no workaround for this problem, therefore it is not recommended that you mix local and global transactions. Case 10880792 has been opened to resolve this problem.

Sybase does not throw an exception when an incorrect database name is specified

Verify that your database name is correctly entered on the data source properties.

Most databases (DB2, Oracle, Informix , MS SQL Server and Cloudscape) throw an exception when the database specified does not exist. But Sybase does not throw an exception when an incorrect database name is specified. Sybase generates an SQL warning and then connects to the default database. If you

misspell the requested database name, Sybase connects you to the master or the default database where the table you requested is not found.

```
Back to Databases-Sybase
Back to Known problems and workarounds
```

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

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

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 55. If you are still unable to resolve the problem, see “Obtaining help from IBM” on page 155.

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.

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

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.

Naming services component 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.
- Open the Log Analyzer on the service log of the server which is hosting the resource you are trying to access and use it to browse error and warning messages.
- With WebSphere Application Server running, run the **dumpNameSpace** command for Windows systems, or the **dumpNameSpace.sh** command for UNIX systems, and pipe, redirect, or ″more″ the output so that it is easily viewed. This command results in a display of objects in the WebSphere Application Server namespace, including the directory path and object name.

  Note: 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 run-time behavior of the WebSphere Application Server Naming service by enabling trace on the following components and reviewing the output:
  - com.ibm.ws.naming.*
  - com.ibm.websphere.naming.*
- If you see an exception that appears to be CORBA related (″CORBA″ appears as part of the exception name) look for a naming-services-specific CORBA minor code, further down in the exception stack, for information on the real cause of the problem. For a list of naming service exceptions and explanations, see the class com.ibm.websphere.naming.WsnCorbaMinorCodes in the Javadoc.

If none of these steps solve the problem:
- For specific problems that can cause access to named object hosted in WebSphere Application Server to fail, see Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP 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.
- 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.

Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP file, or other client

To resolve problems encountered when a servlet, JSP file, stand-alone application or other client attempts to access an enterprise bean, ConnectionPool, or other named object hosted by WebSphere Application Server, you must first verify that the target server can be accessed from the client:
- From a command prompt on the client's server, enter ″`ping` *`server_name`*″ and verify connectivity.
- Use the WebSphere Application Server administrative console to verify that the target resource's application server and, if applicable, EJB module or Web module, is started.

Continue only if there is no problem with connectivity and the target resource appears to be running.

What kind of error are you seeing?
- NameNotFoundException from JNDI lookup operation
- CannotInstantiateObjectException from JNDI lookup operation
- Message NMSV0610I appears in the server's log file, indicating that some Naming exception has occurred
- OperationNotSupportedException from JNDI Context operation.
- ″WSVR0046E: Failed to bind″ error, with Original exception: ″org.omg.CosNaming.NamingContextPackage.AlreadyBound″.
- ConfigurationException from ″new InitialContext″ operation or from a JNDI Context operation with a URL name.
- ServiceUnavailableException from ″new InitialContext″ operation.
- CommunicationException thrown from a ″new InitialContext″ operation.
- NMSV0605E: A Reference object looked up from the context...

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

NameNotFoundException from JNDI lookup operation

If you encounter this exception in trying to access an enterprise bean, data source, messaging resource, or other resource:
- Browse the properties of the target object in the administrative console, and verify that the jndi name it specifies matches the JNDI name the client is using.
- If you are looking up an object that resides on a server different from the one from which 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 may get this error if the object is being hosted on a different server.

– If access is from a stand-alone client, it may be that the object you are attempting access is on a server different from the server from which you obtained the initial context.

To correct this problem, use the fully-qualified JNDIname:
- If the object is in a single server: `cell/nodes/nodeName/servers/serverName/jndiName`. Objects are not supported in this release.
- If the object is on a server cluster: `cell/clusters/clusterName/jndiName`.

CannotInstantiateObjectException from JNDI lookup operation

If you encounter this exception in trying to access an enterprise bean, data source, messaging resource, or other resource, possible causes include:
- A serialized Java object is being looked up, but the necessary classes required to deserialize it are not in the runtime environment.
- A Reference object is being looked up, and the associated factory used to process it as part of the lookup processing is failing.

To determine the precise cause of the problem:
- Look in the JVM logs of the server hosting the target resource. Look for exceptions immediately preceding the CannotInstantiateObjectException. If it is a java.lang.NoClassDefFoundError or java.lang.ClassNotFoundException, make sure the class referenced in the error message can be located by the class loader.
- Print out the stack trace for the root cause and look for the factory class. It will be called by `javax.naming.NamingManager.getObjectInstance()`. The reason for the failure will depend on the factory implementation, and may require you to contact the developer of the factory class.

Message NMSV0610I appears in the server's log file, indicating that some Naming exception has occurred

This error is informational only and is provided in case the exception is related to an actual problem. Most of the time, it is not. If it is, the log file should contain adjacent entries to provide context.
- If no problems are being experienced, ignore this message. Also ignore the message if the problem you are experiencing does not appear to be related to the exception being reported and if there are no other adjacent error messages in the log.
- If a problem is being experienced, look in the log for underlying error messages.
- The information provided in message NMSV0610I can provide valuable debug data for other adjacent error messages posted in response to the Naming exception that occurred.

OperationNotSupportedException from JNDI Context operation

This error has two possible causes:
- An update operation, such as a bind, is being performed with a name that starts with `"java:comp/env"`. This context and its subcontexts are read-only contexts.
- A Context bind or rebind operation of a non-CORBA object is being performed on a remote name space that does not belong to WebSphere Application Server. Only CORBA objects can be bound to these CosNaming name spaces.

To determine which of these errors is causing the problem, check the full exception message.

WSVR0046E: Failed to bind, ejb/jndiName: ejb/jndiName. Original exception : org.omg.CosNaming.NamingContextPackage.AlreadyBound

This error occurs two enterprise bean server applications were installed on the same server such that a binding name conflict occurred. That is, a jndiName value is the same in the two applications' deployment descriptors. The error will surface during server startup when the second application using that jndiName value is started.

To verify that this is the problem, examine the deployment descriptors for all enterprise bean server applications running in the server in search for a jndiName that is specified in more than one enterprise bean application.

To correct the problem, change any duplicate jndiName values to ensure that each enterprise bean in the server process is bound with a different name.

ConfigurationException from "new InitialContext" operation or from a JNDI Context operation with a URL name

If you are attempting to obtain an initial JNDI context, a configuration exception can occur because an invalid JNDI property value was passed to the InitialContext constructor. This includes JNDI properties set in the System properties or in some jndi.properties file visible to the class loader in effect. A malformed provider URL is the most likely property to be incorrect. If the JNDI client is being run as a thin client such that the `CLASSPATH` is set to include all of the individual jar files required, make sure the `.jar` file containing the properties file `com/ibm/websphere/naming/jndiprovider.properties` is in the `CLASSPATH`.

If the exception is occurring from a JNDI Context call with a name in the form of a URL, the current JNDI configuration may not be set up properly so that the required factory class name cannot be determined, or the factory may not be visible to the class loader currently in effect. If the name is a Java: URL, the JNDI client must be running in a J2EE client or server environment. That is, the client must be running in a container.

Check the exception message to verify the cause.

If the exception is being thrown from the InitialContext constructor, correct the property setting or the `CLASSPATH`.

If the exception is being thrown from a JNDI Context method, make sure the property `java.naming.factory.url.pkgs` includes the package name for the factory required for the URL scheme in the name. URL names with the Java scheme can only be used while running in a container.

ServiceUnavailableException from "new InitialContext" operation

This exception indicates that some unexpected problem occurred while attempting to contact the name server to obtain an initial context. The ServiceUnavailableException, like all NamingException objects, can be queried for a root cause. Check the root cause for more information. It is possible that some of the problems described for CommunicationExceptions may also result in a ServiceUnavailableException.

Since this exception is triggered by an unexpected error, there is no probable cause to confirm. If the root cause exception does not indicate what the probable cause is, investigate the possible causes listed for CommunicationExceptions.

CommunicationException thrown from a "new InitialContext" operation

The name server identified by the provider URL cannot be contacted to obtain the initial JNDI context. There are many possible causes for this problem, including:
- The host name or port in the provider URL is incorrect.
- The host name cannot be resolved into an IP address by the domain name server, or the IP address does not match the IP address which the server is actually running under.
- A firewall on the client or server is preventing the port specified in the provider URL from being used.

To correct this problem:
- Make sure the provider URL and the network configurations on the client and server machines are correct.

- Make sure the host name can be resolved into an IP address which can be reached by the client machine. You can do this using the ping command.
- If you are running a firewall, make sure that use of the port specified in the provider URL will be allowed.

Object request broker component troubleshooting tips

This article describes how to diagnose problems related to the WebSphere Application Server Object Request Broker (ORB) component by explaining:
- How to enable tracing for the ORB component.
- What log files to examine for more information.
- Information on the Java packages containing the ORB Service.
- ORB-related tools.
- Where to find configurable settings.
- A listing of CORBA minor codes generated by this component.

Enabling tracing for the Object Request Broker component

The object request broker (ORB) service is one of the WebSphere Application Server run time services. Tracing of messages sent and received by the ORB is a useful starting point for troubleshooting the ORB service. You can selectively enable or disable tracing of ORB messages for each server in a WebSphere Application Server installation, and for each application client.

This tracing is referred to by WebSphere Application Server support as a *comm trace*, and is different from the general purpose trace facility. The trace facility, which shows the detailed run-time behavior of product components, may be used alongside comm trace for other product components, or for the ORB component. The trace string associated with the ORB service is `"ORBRas=all=enabled"`.

You can enable and disable comm tracing using the administrative console or by manually editing the **`server.xml`** file for the server be trace. You must stop and restart the server for the configuration change to take effect.

For example, using the administrative console:
- Navigate to the desired server by clicking **Servers > 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: *`install_dir`*`/config/cells/`*`nodename`*`/nodes/`*`nodename`*`/servers/`*`servername`*`/server.xml`.
- Locate the services entry for the ORB service (`xmi:type="orb:ObjectRequestBroker"`) and set **`commTraceEnabled="true".`**

To enable ORB tracing for client applications, you must specify two ORB properties in the command line used to launch the client application:
- If you are using the WebSphere Application Server launcher, launchClient, use the option **-CCD** or
- If you are using the **java** command directly, use the -D option to specify these parameters:
  - `com.ibm.CORBA.Debug=true`
  - `com.ibm.CORBA.CommTrace=true`

ORB tracing output for thin clients can be directed by setting com.ibm.CORBA.Debug.Output = debugOutputFilename parameter in the command line.

Log files and messages associated with Object Request Broker

Messages and trace information for the ORB are captured primarily in two logs:
- The *`install_dir`*`/logs/`*`servername`*`/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 *install_dir*/logs/*servername*/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]

If tracing of the Object Adapter has been enabled (com.ibm.ejs.oa.*=all=enabled), the following message in the trace.log indicates that the ORB service has started successfully:

EJSORBImpl < initializeORB

The ORB service is one of the first services started during the WebSphere Application Server initialization process. If it is not properly configured, other components such as naming, security, and node agent, are not likely to start successfully. This is obvious in the JVM logs or trace.log of the affected application server.

Java packages containing the Object Request Broker service

The ORB service resides in the following Java packages:
- com.ibm.com.CORBA.*
- com.ibm.rmi.*
- com.ibm.ws.orb.*
- com.ibm.ws.orbimpl.*
- org.omg.CORBA.*
- javax.rmi.CORBA.*

JAR files that contain the previously mentioned packages include:
- *install_dir*/java/jre/lib/ext/ibmorb.jar
- *install_dir*/java/jre/lib/ext/iwsorbutil.jar
- *install_dir*/lib/iwsorb.jar

Tools used with Object Request Broker

The tools used to compile Java remote interfaces to generate language bindings used by the ORB at runtime reside in the following Java packages:
- com.ibm.tools.rmic.*
- com.ibm.idl.*

The JAR file that contains the packages is *install_dir*/java/lib/ibmtools.jar.

Object Request Broker properties

The ORB service requires a number of ORB properties for correct operation. It is not necessary for most users to modify these properties, and it is recommended that only your system administrator modify them when required.. Consult IBM Support personnel for assistance. The properties reside in the orb.properties file, located at *install_dir*/java/jre/lib/orb.properties.

CORBA minor codes

The CORBA specification defines standard minor exception codes for use by the ORB when a system exception is thrown. In addition, the object management group (OMG) assigns each vendor a unique prefix value for use in vendor-proprietary minor exception codes. 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 1. 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 2. 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

Table 2. Decimal minor exception codes 1229123841 to 1229124249 (continued)

1229123970	0x4942f182	BAD_OPERATION_EXTRACT_LONG
1229123971	0x4942f183	BAD_OPERATION_EXTRACT_USHORT
1229123972	0x4942f184	BAD_OPERATION_EXTRACT_ULONG
1229123973	0x4942f185	BAD_OPERATION_EXTRACT_FLOAT
1229123974	0x4942f186	BAD_OPERATION_EXTRACT_DOUBLE
1229123975	0x4942f187	BAD_OPERATION_EXTRACT_LONGLONG
1229123976	0x4942f188	BAD_OPERATION_EXTRACT_ULONGLONG
1229123977	0x4942f189	BAD_OPERATION_EXTRACT_BOOLEAN
1229123978	0x4942f18a	BAD_OPERATION_EXTRACT_CHAR
1229123979	0x4942f18b	BAD_OPERATION_EXTRACT_OCTET
1229123980	0x4942f18c	BAD_OPERATION_EXTRACT_WCHAR
1229123981	0x4942f18d	BAD_OPERATION_EXTRACT_STRING
1229123982	0x4942f18e	BAD_OPERATION_EXTRACT_WSTRING
1229123983	0x4942f18f	BAD_OPERATION_EXTRACT_ANY
1229123984	0x4942f190	BAD_OPERATION_INSERT_OBJECT_1
1229123985	0x4942f191	BAD_OPERATION_INSERT_OBJECT_2
1229123986	0x4942f192	BAD_OPERATION_EXTRACT_OBJECT_1
1229123987	0x4942f193	BAD_OPERATION_EXTRACT_OBJECT_2
1229123988	0x4942f194	BAD_OPERATION_EXTRACT_TYPECODE
1229123989	0x4942f195	BAD_OPERATION_EXTRACT_PRINCIPAL
1229123990	0x4942f196	BAD_OPERATION_EXTRACT_VALUE
1229123991	0x4942f197	BAD_OPERATION_GET_PRIMITIVE_TC_1
1229123992	0x4942f198	BAD_OPERATION_GET_PRIMITIVE_TC_2
1229123993	0x4942f199	BAD_OPERATION_INVOKE_NULL_PARAM_1
1229123994	0x4942f19a	BAD_OPERATION_INVOKE_NULL_PARAM_2
1229123995	0x4942f19b	BAD_OPERATION_INVOKE_DEFAULT_1
1229123996	0x4942f19c	BAD_OPERATION_INVOKE_DEFAULT_2
1229123997	0x4942f19d	BAD_OPERATION_UNKNOWN_BOOTSTRAP_METHOD
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

Table 2. Decimal minor exception codes 1229123841 to 1229124249 (continued)

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
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 3. 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

Table 3. Decimal minor exception codes 1229124250 to 1229125764 (continued)

1229124376	0x4942f318	GET_PROPERTIES_ERROR
1229124384	0x4942f320	BOOTSTRAP_SERVER_NOT_AVAIL
1229124392	0x4942f328	INVOKE_ERROR
1229124481	0x4942f381	BAD_HEX_DIGIT
1229124482	0x4942f382	BAD_STRINGIFIED_IOR_LEN
1229124483	0x4942f383	BAD_STRINGIFIED_IOR
1229124485	0x4942f385	BAD_MODIFIER_1
1229124486	0x4942f386	BAD_MODIFIER_2
1229124488	0x4942f388	CODESET_INCOMPATIBLE
1229124490	0x4942f38a	LONG_DOUBLE_NOT_IMPLEMENTED_1
1229124491	0x4942f38b	LONG_DOUBLE_NOT_IMPLEMENTED_2
1229124492	0x4942f38c	LONG_DOUBLE_NOT_IMPLEMENTED_3
1229124496	0x4942f390	COMPLEX_TYPES_NOT_IMPLEMENTED
1229124497	0x4942f391	VALUE_BOX_NOT_IMPLEMENTED
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

Table 3. Decimal minor exception codes 1229124250 to 1229125764 (continued)

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
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 4. 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

Table 4. Decimal minor exception codes 1299125765 to 1229125906 (continued)

1229125772	0x4942f88c	UNSPECIFIED_MARSHAL_11
1229125773	0x4942f88d	UNSPECIFIED_MARSHAL_12
1229125774	0x4942f88e	UNSPECIFIED_MARSHAL_13
1229125775	0x4942f88f	UNSPECIFIED_MARSHAL_14
1229125776	0x4942f890	UNSPECIFIED_MARSHAL_15
1229125777	0x4942f891	UNSPECIFIED_MARSHAL_16
1229125778	0x4942f892	UNSPECIFIED_MARSHAL_17
1229125779	0x4942f893	UNSPECIFIED_MARSHAL_18
1229125780	0x4942f894	UNSPECIFIED_MARSHAL_19
1229125781	0x4942f895	UNSPECIFIED_MARSHAL_20
1229125782	0x4942f896	UNSPECIFIED_MARSHAL_21
1229125783	0x4942f897	UNSPECIFIED_MARSHAL_22
1229125784	0x4942f898	UNSPECIFIED_MARSHAL_23
1229125785	0x4942f899	UNSPECIFIED_MARSHAL_24
1229125786	0x4942f89a	UNSPECIFIED_MARSHAL_25
1229125787	0x4942f89b	UNSPECIFIED_MARSHAL_26
1229125788	0x4942f89c	UNSPECIFIED_MARSHAL_27
1229125789	0x4942f89d	UNSPECIFIED_MARSHAL_28
1229125790	0x4942f89e	UNSPECIFIED_MARSHAL_29
1229125791	0x4942f89f	UNSPECIFIED_MARSHAL_30
1229125792	0x4942f8a0	UNSPECIFIED_MARSHAL_31
1229125793	0x4942f8a1	UNSPECIFIED_MARSHAL_32
1229125794	0x4942f8a2	UNSPECIFIED_MARSHAL_33
1229125795	0x4942f8a3	UNSPECIFIED_MARSHAL_34
1229125796	0x4942f8a4	UNSPECIFIED_MARSHAL_35
1229125797	0x4942f8a5	UNSPECIFIED_MARSHAL_36
1229125798	0x4942f8a6	UNSPECIFIED_MARSHAL_37
1229125799	0x4942f8a7	UNSPECIFIED_MARSHAL_38
1229125800	0x4942f8a8	UNSPECIFIED_MARSHAL_39
1229125801	0x4942f8a9	UNSPECIFIED_MARSHAL_40
1229125802	0x4942f8aa	UNSPECIFIED_MARSHAL_41
1229125803	0x4942f8ab	UNSPECIFIED_MARSHAL_42
1229125804	0x4942f8ac	UNSPECIFIED_MARSHAL_43
1229125805	0x4942f8ad	UNSPECIFIED_MARSHAL_44
1229125806	0x4942f8ae	UNSPECIFIED_MARSHAL_45
1229125807	0x4942f8af	UNSPECIFIED_MARSHAL_46
1229125808	0x4942f8b0	UNSPECIFIED_MARSHAL_47
1229125809	0x4942f8b1	UNSPECIFIED_MARSHAL_48
1229125810	0x4942f8b2	UNSPECIFIED_MARSHAL_49
1229125811	0x4942f8b3	UNSPECIFIED_MARSHAL_50
1229125812	0x4942f8b4	UNSPECIFIED_MARSHAL_51
1229125813	0x4942f8b5	UNSPECIFIED_MARSHAL_52
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

Table 4. Decimal minor exception codes 1299125765 to 1229125906 (continued)

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
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 5. 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

Table 5. Decimal minor exception codes 1229125907 to 1229126567 (continued)

1229125917	0x4942f91d	SET_ONE_VALUE_NOTIMPLEMENTED
1229125918	0x4942f91e	SET_VALUES_NOTIMPLEMENTED
1229125919	0x4942f91f	DELETE_VALUES_NOTIMPLEMENTED
1229125920	0x4942f920	GET_VALUES_NOTIMPLEMENTED
1229125922	0x4942f922	GET_CURRENT_NOTIMPLEMENTED_1
1229125923	0x4942f923	GET_CURRENT_NOTIMPLEMENTED_2
1229125924	0x4942f924	CREATE_OPERATION_LIST_NOTIMPLEMENTED_1
1229125925	0x4942f925	CREATE_OPERATION_LIST_NOTIMPLEMENTED_2
1229125926	0x4942f926	GET_DEFAULT_CONTEXT_NOTIMPLEMENTED_1
1229125927	0x4942f927	GET_DEFAULT_CONTEXT_NOTIMPLEMENTED_2
1229125928	0x4942f928	SHUTDOWN_NOTIMPLEMENTED
1229125929	0x4942f929	WORK_PENDING_NOTIMPLEMENTED
1229125930	0x4942f92a	PERFORM_WORK_NOTIMPLEMENTED
1229125931	0x4942f92b	COPY_TK_ABSTRACT_NOTIMPLEMENTED
1229125932	0x4942f92c	PI_CLIENT_GET_POLICY_NOTIMPLEMENTED
1229125933	0x4942f92d	PI_SERVER_GET_POLICY_NOTIMPLEMENTED
1229125934	0x4942f92e	ADDRESSING_MODE_NOTIMPLEMENTED_1
1229125935	0x4942f92f	ADDRESSING_MODE_NOTIMPLEMENTED_2
1229125936	0x4942f930	SET_OBJECT_RESOLVER_NOTIMPLEMENTED
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

Table 5. Decimal minor exception codes 1229125907 to 1229126567 (continued)

1229126041	0x4942f999	MARSHAL_NO_MEMORY_25
1229126042	0x4942f99a	MARSHAL_NO_MEMORY_26
1229126043	0x4942f99b	MARSHAL_NO_MEMORY_27
1229126044	0x4942f99c	MARSHAL_NO_MEMORY_28
1229126045	0x4942f99d	MARSHAL_NO_MEMORY_29
1229126046	0x4942f99e	MARSHAL_NO_MEMORY_30
1229126047	0x4942f99f	MARSHAL_NO_MEMORY_31
1229126401	0x4942fb01	RESPONSE_TIMED_OUT
1229126402	0x4942fb02	FRAGMENT_TIMED_OUT
1229126529	0x4942fb81	NO_SERVER_SC_IN_DISPATCH
1229126530	0x4942fb82	NO_SERVER_SC_IN_LOOKUP
1229126531	0x4942fb83	NO_SERVER_SC_IN_CREATE_DEFAULT_SERVER
1229126532	0x4942fb84	NO_SERVER_SC_IN_SETUP
1229126533	0x4942fb85	NO_SERVER_SC_IN_LOCATE
1229126534	0x4942fb86	NO_SERVER_SC_IN_DISCONNECT
1229126539	0x4942fb8b	ORB_CONNECT_ERROR_1
1229126540	0x4942fb8c	ORB_CONNECT_ERROR_2
1229126541	0x4942fb8d	ORB_CONNECT_ERROR_3
1229126542	0x4942fb8e	ORB_CONNECT_ERROR_4
1229126543	0x4942fb8f	ORB_CONNECT_ERROR_5
1229126544	0x4942fb90	ORB_CONNECT_ERROR_6
1229126545	0x4942fb91	ORB_CONNECT_ERROR_7
1229126546	0x4942fb92	ORB_CONNECT_ERROR_8
1229126547	0x4942fb93	ORB_CONNECT_ERROR_9
1229126548	0x4942fb94	ORB_REGISTER_1
1229126549	0x4942fb95	ORB_REGISTER_2
1229126553	0x4942fb99	ORB_REGISTER_LOCAL_1
1229126554	0x4942fb9a	ORB_REGISTER_LOCAL_2
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 6. 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

Table 6. Decimal minor exception codes 1229126568 to 1330446377 (continued)

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
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

Table 6. Decimal minor exception codes 1229126568 to 1330446377 (continued)

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
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). If you do not 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.

Messaging component troubleshooting tips

Problems deploying or executing applications which use the WebSphere Application Server messaging capabilities

If you are having problems deploying or executing applications which use the WebSphere Application Server messaging capabilities, review these articles in the WebSphere Application Server information center:

- “Troubleshooting WebSphere messaging” on page 219
- “Troubleshooting message-driven beans” on page 221
- “Troubleshooting transactions” on page 263

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 on the uninstall command, see uninstall command.

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.

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?

Legend for ″How do I?...″ links

Documentation	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+

Set traces, logs, and other controls

Use trace to obtain detailed information about the execution of product components, including application servers, clients, and other processes in the environment.

Documentation Show me Tell me

- Console
- Scripting

Work with message logs

WebSphere Application Server can write system messages to several general purpose logs. These include the JVM logs, the process logs and the IBM service log.

Documentation Show me

Detect hung threads

A common error in J2EE applications is a hung thread. The hang detection option 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, the product notifies you so that you can troubleshoot the problem.

Documentation Teach me

Detect product configuration file problems

Use an administrative console page to identify and view problems that exist in the current configuration.

Documentation

Troubleshoot problems that occur during a task

Troubleshoot problems that crop up during a main task such as migrating, installing, administering, securing, or deploying applications.

Documentation

Troubleshoot particular WebSphere application problems

Expand **Troubleshooting > Troubleshooting WebSphere applications** in the information center table of contents for information about troubleshooting specific application components and their administrative configurations (such as EJB container settings).

Debug WebSphere applications during development

In order to debug your application, you must use your application development tool (such as WebSphere Studio Application Developer) to create a Java project or a project with a Java nature. You must then import the program that you want to debug into the project.

Documentation

Add tracing and logging to your applications

Designers and developers of applications that run on the application server might find it useful to use Java logging for generating their application logging. This approach has advantages over simply adding System.out.println statements to your code.

Documentation

Collect troubleshooting details for IBM Support

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.

Documentation

Using JSR47 for logging

The Logging API Specification (or Java logging), which is defined with Java Specification Request (JSR) 47, is the logging toolkit that is provided by the java.util.logging package. Java logging provides a standard logging API for your applications.

Documentation

Using JSR47 for logging: Writing a handler

There may be occasions when you only want to propagate log records to your own log handlers, rather than participate in integrated logging; in other words, to use a stand alone log handler.

Documentation

Using JSR47 for logging: Writing a formatter

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

Documentation

Using JSR47 for logging: Writing a filter

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

Documentation

Using JSR47 for logging: Configuring access logs

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, and specify a format for log output. You can also set a log level for each logger.

Documentation

Using Common Base Events for logging

WebSphere Application Server uses Common Base Events within its logging framework. Common Base Events can be created explicitly and logged via 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.

Documentation

Creating Common Base Events

In cases where the events generated by the Java logging API are insufficient to describe the event that needs to be captured, Common Base Events can be created using 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. 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.

Documentation

Chapter 3. Debugging applications

To debug your application, you must use your application development tool (such as Rational Application Developer) to create a Java project or a project with a Java nature. You must then import the program that you want to debug into the project. By following the steps below, you can import the WebSphere Application Server examples into a Java project.

There are two debugging styles available:

- **Step-by-step** 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 need not 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 then specify a name for the project (such as WASExamples).
4. Press **Finish** to create the project.
5. Select the new project, choose **File > Import > File System**, then **Next** to open the import file system dialog.
6. Select the directory Browse pushbutton and go to the following directory: *installation_root*\installedApps*node_name* \DefaultApplication.ear\DefaultWebApplication.war.
7. Select the checkbox next to DefaultWebApplication.war in the left side of the Import dialog and then click **Finish**. This will import the 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. To do this, select **Properties** from the right-click menu. Choose the Java Build Path node and then select the Libraries tab. Use the Add External JARs pushbutton to add the following JAR files:
   - *installation_root*\installedApps*node_name*\DefaultApplication.ear\Increment.jar.

     Once you have added this JAR file, select it and use the **Attach Source** pushbutton to attach Increment.jar as the source - Increment.jar contains both the source and class files.
   - *installation_root*\lib\j2ee.jar
   - *installation_root*\lib\pagelist.jar
   - *installation_root*\\lib\webcontainer.jar

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

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

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 IBM Support page.

Debugging with the Application Server Toolkit

The Application Server Toolkit, included with the WebSphere Application Server on a separately-installable CD, includes debugging functionality that is built on the Eclipse workbench. 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. To learn more about Log and Trace Analyzer, launch the Application Server Toolkit, and select **Help > Help Contents**. To learn about known limitations and problems that are associated with the Application Server Toolkit, see the Application Server Toolkit release notes.

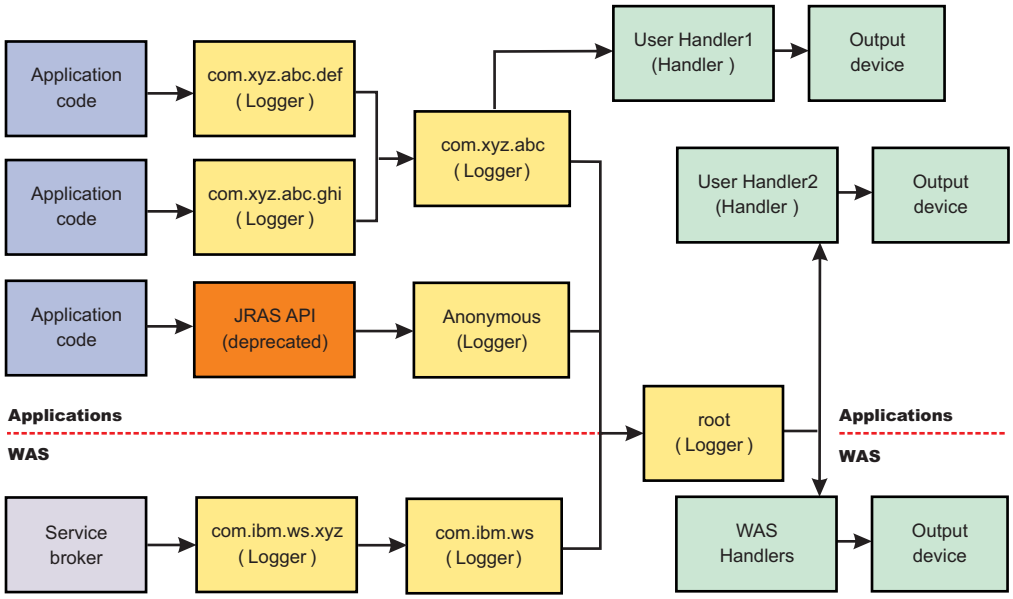
Chapter 4. Adding logging and tracing to your application

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 simply adding `System.out.println` statements to your code:

- Your messages are displayed in the WebSphere Application Server standard log files, using 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.
- You can view your messages with the Log Analyzer tool.



Logging and tracing with Java logging

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 has 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*.

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.

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, meaning they are displayed in the national language of the end user. Although the destination and lifetime of messages might be configurable, some level of message logging should be enabled in normal system operation. Message logging should be used 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 the following topics:
- "Configuring logging properties using the administrative console" on page 67.

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 has completed startup) or because a potential problem has been detected (such as a timeout waiting for a resource) a message will be 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 logged to it. If the resource bundle is not specified, a logger will use the same resource bundle as its parent.

Loggers can be configured with a level. If specified the level is compared by the logger to incoming events -- events less severe than the level set for the logger are ignored by the logger. If the level is not specified, a logger will take on the level used by its parent. The default level for loggers is Level.INFO.

Loggers can have zero or more attached handlers. If supplied, all events logged to the logger will be passed to the attached handlers (for more details see "Log handlers" on page 65). Handlers write events to output destinations such as log files or network sockets. When a logger has finished passing a logged event to all of the handlers attached to that logger, the logger then passes the event to the handlers attached to the parents of the logger. This process will stop if a parent logger has been 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 up 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 logger ″com.ibm.websphere.ras″ has a parent logger named ″com.ibm.websphere″, which has a parent named ″com.ibm″, and so on. 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 logger ″com″.

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 ResourceBundle) methods add the capability of specifying a ResourceBundle 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 a complete list of methods, see the java.util.logging documentation at http://java.sun.com/j2se/.

Log handlers

Handlers write log record objects to an output device. Some examples of an output device are log files, sockets, and notification mechanisms.

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

Handlers can be configured with a level. The handler compares the level specified in the logged object to the level 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 could decide to first include the timestamp, followed by a string representation of the level, followed by the message included in the logged object. The handler writes this formatted representation to the output device.

Both loggers and handlers can have levels and filters, and an logged object must pass all of these in order to be output. For example, you can set the logger level to FINE, but if the handler level is set at WARNING, then only WARNING level messages will appear 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, then the logger never sends messages lower than WARNING to the log handler.

WebSphere Application Server uses the following set of log handlers that are available to all loggers:

- Diagnostic trace
- JMX notification object
- Service log
- SystemErr
- SystemOut

For instructions on how to configure these log handlers, see “Configuring logging properties using the administrative console” on page 67

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 (see “Configuring logging properties using the administrative console” on page 67. 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 has been 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 needed to debug problems
All	All events are logged. If you have created custom levels, ″All″ would include your custom levels, and could 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 67

Note: Trace information, which are events at levels Fine, Finer and Finest, can only be written 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.

Log filters

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. For a complete list of methods, see the java.util.logging documentation at http://java.sun.com/j2se/

Log formatters

This article describes what a log formatter is.

Handlers may be configured with a formatter, which knows how to format log records. The event (represented by the log record 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.

The formatter is responsible for rendering the event for output. This usually means the formatter uses the ResourceBundle specified in the event to look up the message in the appropriate language.

Formatters are attached to handlers using the setFormatter method.

WebSphere Application Server allows you to configure the formatter to be used with trace, SystemOut.log, and SystemErr.log log files:

- **Basic (Compatible)** - Preserves only basic trace information. The option allows you to minimize the amount of space taken up by the trace output.
- **Advanced** - Preserves more specific trace information. **Advanced** allows you to see detailed trace information for use in troubleshooting and problem determination.
- **Log Analyzer** - Preserves trace information in a format that is compatible with the Log Analyzer tool, so that you can use the trace output as input to the Log Analyzer tool.

You can select a formatter for a handler using the administrative console panels. See “Diagnostic trace service settings” on page 138 for details.

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

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, and specify a format for log output. You can also 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 propagate the change to their children, as necessary.

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

1. Set the output properties for a log:
   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 the name of a system log to configure (Diagnostic Trace, JVM Logs, Process Logs or IBM Service Logs).
   e. To make a static change to the system log configuration, click the **Configuration** tab. To change the configuration dynamically, click the **Runtime** tab.
   f. Change the properties for the selected log according to your needs.
   g. Click **Apply**.

h. Click **OK**.

2. 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. See “Log level settings” for a description of each level.
   g. Click **Apply**.
   h. Click **OK**.
3. To have static configuration changes take effect, stop then restart the Application Server.

Log level settings

Use this page 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

Specifies tracing 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 below. You can enter the log detail level string directly, or generate it using the graphical trace interface.

If you select the configuration tab, a 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 groups is displayed will all such components currently registered on the running server. This list is static; you will not see your application logger names on the runtime tab.

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 * as a wildcard to indicate components that include all classes in all packages contained by the specified component. For example:

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

com.ibm.ws.*
specifies all classes whose package name begins with com.ibm.ws.

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

Note: 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 desired 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 below lists the valid levels for application servers at WebSphere Application Server Version 6 and higher, and the valid logging and trace levels for earlier versions:

Version 6 Logging Level	Logging Level pre-Version 6	Trace Level pre-Version 6	Content / Significance
Off	Off	All disabled*	Logging is turned off. * In Version 6, a trace level of All disabled will turn off trace, but will not turn off logging. Logging will be enabled from the Info level.
Fatal	Fatal	-	Task cannot continue and component/ application/ server cannot function.
Severe	Error	-	Task cannot continue but component/ application/ 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 / 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″ would include those levels, and could provide a more detailed trace than ″finest″.

When you enable a logging level in version 6, you are also enabling all of the levels above it. For example, if you set the logging level to warning on your version 6 application server, then warning, severe and fatal events will be processed.

Note: Trace information, which are events at levels Fine, Finer and Finest, can only be written 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

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

Use this panel to configure an HTTP error log and NCSA access logs for an HTTP transport channel.

The HTTP error log logs HTTP errors. The level of error logging that occurs is dependent on the value selected for the Error log level field.

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

After you have configured the HTTP error log and the NCSA access logs, make sure the Enable NCSA access logging field is selected for the HTTP channels for which you want logging to take place. 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 will be initialized when the server is started.

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

NCSA access log format: Specifies 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 in 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: Specifies the directory path and 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: Specifies the maximum size, in megabytes, of the HTTP error log file. When this size is reached, an archive log named *logfile_name*.1 is created. However, every subsequent time that the original log file overflows this archive file is overwritten with the most current version of the original log file.

Error log level:

Specifies 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 Errors in responses to clients are logged. These errors require Application Server administrator intervention if they are caused by 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 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.

Configuring logging properties for an application

The logger.properties file allows you to set logger attributes for specific loggers. The properties file is loaded the first time the method `Logger.getLogger(`*`loggername`*`)` is called within an application.

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 JAR file for the application
- directories included in the class path of 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 - minimum LogRecord level for which localization will be attempted
  - group - logical group that this component belongs to
  - eventfactory - The Common Base Event template file to use with the event factory. Note that 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 package com.ibm.compXYZ would be called com.ibm.compXYZ.event.xml
- Logger data information
  - product name
  - organization name
  - component name
  - extensions – additional properties

Sample logger.properties file

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

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

Sample security

Purpose

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 to allow the application to programmatically alter controlled properties of loggers and handlers.

```
//////////////////////////////////////////////
//
// 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 article describes how to use Java logging within an application.

To instrument 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 will be used by the application, create a resource bundle as described in “Creating log resource bundles and message files” on page 76.
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

There are various ways in which to log messages or trace using Java logging. The following guidelines will help you to use Java logging to log messages and add tracing:

1. Use level WsLevel 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
     ```

2. Use the `logp` method instead of `log` or `logrb` since `logp` 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.
3. Avoid using the `logrb` method since this leads to inefficient caching of resource bundles which results in poor performance.
4. Use the isLoggable method to avoid creating data for a logging call that will 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
//   - can't specify message substitution parameters
//   - can't specify class/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 - not generally recommended due to lack of class / method names
//   - enable use of WebSphere Application Server specific levels
//   - enable use of message substitution parameters
//   - can't specify class/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 - the recommended way to log
//   - enable use of WebSphere Application Server specific levels
```

```
//   - enable use of message substitution parameters
//   - enable use of class/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 - not generally recommended due to diminished performance of switching resource bundles on the fly
//   - enable use of WebSphere Application Server specific levels
//   - enable use of message substitution parameters
//   - enable use of class/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 - recommended 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 - not generally recommended due to lack of message - use logp with a throwable parameter instead
if (logger.isLoggable(Level.FINER))
 logger.throwing(className, methodName, throwable);


// "Convenience" methods - not generally recommended due to lack of class / method names
//   - can't specify message substitution parameters
//   - can't specify class/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 - not generally recommended due to lack of class / method names
//   - enable use of WebSphere Application Server specific levels
//   - enable use of message substitution parameters
//   - can't specify class/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 - the recommended way to log
//   - enable use of WebSphere Application Server specific levels
//   - enable use of message substitution parameters
//   - enable use of class/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 - not applicable for trace logging since no localization is involved
```

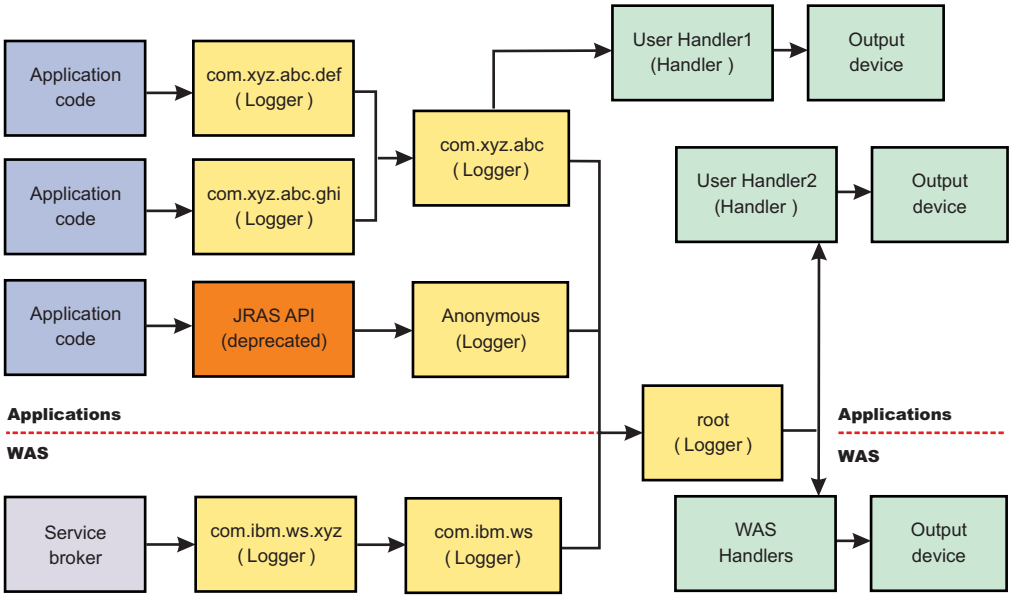
Understanding the logger hierarchy

WebSphere Application Server handlers are attached to the Java root logger in the logger hierarchy. As a result, any request from anywhere in the logger tree can be processed by WebSphere Application Server handlers. WebSphere Application Server handlers will process both standard log records and WebSphere Application Server log records. You can configure the system to:

- Forward all application logging requests to the WebSphere Application Server handlers. This is the default behavior.

- Forward all application logging requests to your own custom handlers. To do this, set **useParentHandlers** 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. To do this, set **useParentHandlers** to true on one of your non-root custom loggers (true is the default setting) , and then attach your handlers to that logger.
- 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 example below shows how these requirements can be met using the Java logging infrastructure.



Creating log resource bundles and message files

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

The `PropertyResourceBundle` is the preferred mechanism to use.

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

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

By default, the WebSphere Application Server runtime localizes all the messages when they are logged. This 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 are logged. 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's 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 only white space, 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 only of white space, denotes a single property. A backslash (`\`) is treated as the line-continuation character.
   - The syntax for a property file consists of a key, a separator, and an element. Valid separators include the equal sign (`=`), colon (`:`), and white space ( ).
   - The key consists of all characters on the line from the first non-white space character to the first separator. Separator characters can be included in the key by escaping them with a backslash (`\`), but doing this is not recommended, because escaping characters is error prone and confusing. It is instead recommended that you use a valid separator character that does not appear in any keys in the properties file.
   - White space after the key and separator is ignored until the first non-white space character is encountered. All characters remaining before the line-termination character define the element.

   See the Java documentation for the `java.util.Properties` class for a full description of the syntax and construction of properties files.
2. The file can then be translated into localized versions of the file with language-specific file names (for example, a file named `DefaultMessages.properties` can be translated into DefaultMessages_de.properties for German and DefaultMessages_ja.properties for Japanese).
3. When the translated resource bundles are available, put the bundle in a directory that is part of the application's classpath.
4. When a message logger is obtained from the log manager, it can be configured to use a particular resource bundle. Messages logged via the `Logger()` API will use this resource bundle when message localization is performed. At run time, the user's locale setting is used to determine the properties file from which to extract the message specified by a message key, thus ensuring that the message is delivered in the correct language.
5. 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's location relative to any directory in the classpath. For instance, if the property resource bundle named DefaultMessages.properties is located in the *baseDir*`/`*subDir1*`/`*subDir2*`/resources` directory and *baseDir* is in the class path, the name *subdir1*`.`*subdir2*`.resources.DefaultMessage` is passed to the message logger to identify the resource bundle.

Developing log resource bundles:
Resource bundle sample

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

For this sample, four localizable messages are provided. The properties file is created and the key-value pairs inserted into it. All the normal properties files conventions and rules apply to this file. In addition, the creator must be aware of other restrictions imposed on the values by the Java `MessageFormat` class. For example, apostrophes must be ″escaped″ or they will cause a problem. Also avoid use of non-portable

characters. WebSphere Application Server does not support usage of extended formatting conventions that the MessageFormat class supports, such as {1, date} or {0,number, integer}.

Assume that the base directory for the application that uses this resource bundle is "baseDir" and that this directory will be in the classpath. Assume that the properties file is stored in a subdirectory of baseDir that is not in the classpath (e.g. baseDir/subDir1/subDir2/resources). In order to allow the messages file to be resolved, the name subDir1.subDir2.resources.DefaultMessage is used to identify the PropertyResourceBundle and is passed to the message logger.

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

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

Once the file DefaultMessages.properties is created, the file can be sent to a translation center where the localized versions will be generated.

Creating a custom log handler

There may be occasions when you only 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 provided by the IBM Developer Java Technology Edition. If you are not familiar with handlers as implemented by the Developer's Kit, you can get more information from various texts, or by reading the Javadoc for java.util.logging.

The following is a sample of 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-Javadoc)
```

```
  * @see java.util.logging.Handler#publish(java.util.logging.LogRecord)
  */
 public void publish(LogRecord record) {
  // ensure that this LogRecord should be logged by this Handler
  if (!isLoggable(record))
   return;

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

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

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

Creating a custom filter

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

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

The following is an example of a custom filter:

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

/**
 * MyCustomFilter rejects any LogRecords 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-Javadoc)
  * @see java.util.logging.Filter#isLoggable(java.util.logging.LogRecord)
  */
 public boolean isLoggable(LogRecord record) {
  return (acceptableLevels.contains(record.getLevel()));
 }

}
```

Creating a custom formatter

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

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

The following is an example of 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();
 }
}
```

Using custom handlers, filters, and formatters

In some cases you may wish 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 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"Understanding the logger hierarchy" on page 75). 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 which we wish 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 executed, 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 has been 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
```

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 should be recorded for later analysis or which may require additional work to be triggered. An *event* is a structured notification that reports information related to a situation. An event reports three kinds of information:

- The situation itself (what has happened)
- The identity of the affected component (for example, the server that has shut down)
- The identity of the component that is reporting the situation (which might be the same as the affected component)

The application creating 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 a XML document 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 defining a common representation of events that is intended for use by enterprise management and business applications. This standard, developed by the IBM Autonomic Computing Architecture Board, supports encoding of logging, tracing, management, and business events using a common XML-based format, making 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 should contain all of the information needed by the consumers to understand the event. This includes information 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 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

This topic describes types of problem determination events.

Problem determination involves using 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 may 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 operations. 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.

Common Base Events are primarily used to represent log events.

The structure of the Common Base Event

A Common Base Event contains several structural elements. These are:

- 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 fields in the Common Base Event and their usage requirements for problem determination events. It 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

Note:

1. 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.
2. Some of the elements are optional, but when included, they include sub-elements and attributes that are required. For example, reporterComponentId is of type ComponentIdentification. The component attribute in ComponentIdentification is required. Therefore, the reporterComponentId.component attribute is required, but only when the parent element (reporterComponentId) is included.
3. 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.
4. The contextDataElements element can be included multiple times to supply context data information.
5. The associatedEvents element can be included multiple times to supply correlation data. There are no recommended uses of this element for the producers of problem determination data, and it is in fact discouraged.

Common Header Information

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

- the version of this Common Base Event (version).
- the date and time when the event was generated (creationTime).
- the severity of the condition (situation) identified by the event (severity and priority).

- the type of event that was captured (extensionName).
- identifiers that can be used to quickly identify a specific event within a set of events (localInstanceId and globalInstanceId).
- information that allows a system to efficiently report multiple events of the same type, by consolidating those events into a single event (repeatCount and elapsedTime).
- sequence information that allows a system to order a set of events in other ways than time of capture (sequenceNumber).

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 article will provide additional information about how to format and use these fields for problem determination events, which should be used to clarify and extend the information provided in the other documents.

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 is more restrictive than the base specification for the Common Base Event, which lists this as an optional field because effective and efficient problem determination requires the ability to quickly identify the information needed to resolve a problem as well as prioritize the problems that need to be addressed. Typically the following values are used for problem determination events:

10	Information	Log information events (normal conditions, events 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 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 was able to correct. These events may require administrator action or intervention.
40	Minor	Minor errors describe events that represent an unrecoverable error within a component. The failure affects the component's ability to service some requests. The business application is able to continue to perform its normal functions, but its overall operation may 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's ability to service most requests. The business application is able to continue most (but not all) of its normal functions and its overall operation may 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 may be able to continue some normal functions, but its overall operation may be degraded. These events require administrator action or intervention to address the condition.

msg
Refer to "Message Data" on page 90 for information on this attribute.

priority
The usage 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. When the priority field is used, it should only be used to enhance the information provided in severity field, i.e. prioritize events of the same severity.

extensionName
The extensionName field is used to communicate the 'type' of event being reported, for example, what general class of events is being reported. In many cases this field provides an indication of what additional data should be expected to be supplied 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/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/consolidation by event management and analysis systems.

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

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

Component Identification (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 described by the event (the sourceComponentId) and which component emitted the event (the reporterComponentId). Typically, these are the same component, in which case only the sourceComponentId is supplied. Some notes and scenarios on when these two elements in the Common Base Event should be used:

- the sourceComponentId is always used to identify the component experiencing the condition described by the event.
- the reporterComponentId is used to identify the component that actually produced and emitted the event. This element is typically only used within events 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 captured by a component into Common Base Event format. The event conversion function is considered an extension of the component and should NOT be identified separately.

The information 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 (i.e. subcomponent) of a component, e.g. 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, i.e. 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, i.e. the 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 (e.g. the operating system name for a software application, the application server name for a 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 will provide additional information about how to format and use some of these fields for problem determination events, which should be used to clarify and extend the information provided in the other documents.

Component

The component field in a problem determination event is used to identify the manageable asset 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#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 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, i.e. 'What part failed?'. Examples of typical subcomponents and their names are:

- Intel Pentium processor within a server system ("Intel Pentium IV Processor")
- the EJB container within a web application server ("EJB 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 it is recommended that the convention shown above for naming a Java class or the combination of a Java class and method is followed. Subcomponent is a required field in the Common Base Event.

componentIdType

The componentIdType field is required by the base Common Base Event specification, but provides minimal value for problem determination events. The only recommendation regarding this field for problem determination events is to discourage the use of the "application" value. 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 it should be provided within problem determination events whenever it is available. The only reason this is not a required field for problem determination events is there are instances where the issuing component may 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 it should be provided within problem determination events whenever it is available.

The instanceID should always be provided when a software component is being identified and should identify the operational instance of the component (for example, which operation instance of an installed software image is actually associated with the event). This value should also be provided for hardware components, but not all hardware components support the concept of operational instances.

The format of the supplied value is defined by the component, but must be a value that can be used by an analysis system (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 EJB
- **serial number** for a hardware processor

processId
The processId field is listed as an optional value within the Common Base Event specification, but it should be provided for problem determination events whenever it is available and applicable. It should always be provided for software-generated events, and should identify the operating system process associated with the component identified in the event. The format of the thread ID should match the format of the operating system (or other execution environment, such as a Java Virtual Machine). This field is typically not applicable or used for events emitted by hardware (for example, firmware).

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

executionEnvironment

The executionEnvironment field, when used, should identify the immediate execution environment 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 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 EJB.

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 being 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, along with the values and formats used to describe these situations. The Common Base Event Developer's Guide [CBEBASE] provides general usage guidelines.

There are a few recommendations for situation information for problem determination events, such as:

- Whenever possible, use the situation categorizations and qualifiers 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 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 (i.e. an event with severity=warning) typically indicates a recoverable failure, but the situation settings can be interpreted as unrecoverable failures (e.g. ConnectSituation, successDisposition=UNSUCCESSFUL). The appropriate situation categorization should always be used and the severity setting will indicate the severity of the situation, i.e. 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 describing the specific event being reported within the msg field of the Common Base Event. The text associated with events representing actual messages or log entries is expected to be internationalized (in other words, translated and localized). We strongly recommend including 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 was created and how it should be interpreted, and is particularly invaluable when trying to interpret the event programmatically or when trying to interpret the message independent of the locale or language used to format the message text.

Note: Readers of this section of the document should be familiar with the concepts associated with creating internationalized messages (in other words, messages that are translated and localized). A good source of education on these concepts is provided by the documentation 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 text (i.e. the value of the msg field) provided with an event:

- The locale of the supplied message text, which identifies how the locale-independent fields within the message were formatted, as well as the language of the message (msgLocale).
- A locale-independent identifier associated with the message that can be used to interpret the message independent of the message language, message locale, and how the message was formatted (msgId and msgIdType).
- Information on how a translated message was created, including:
  - The identifier used to retrieve the message template (msgCatalogId).
  - The name and type of message catalog used to retrieve the message template (msgCatalog and msgCatalogType).
  - Any locale-independent information that was 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 will provide additional information about how to format and use these fields for problem determination events, which should be used to clarify and extend the information provided in the other documents.

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 is more restrictive than the base specification for the Common Base Event, which lists this as an optional field; because effective and efficient problem determination requires the ability to quickly identify the condition being reported by the event. The format and usage of this message is component-specific, but the following general guidelines should be followed:

- The message text supplied with messages and log events is expected to be internationalized.
- The locale of the supplied message text should be provided using the msgLocale field in the msgDataElement element of the Common Base Event.
- Additional information regarding the format and construction of internationalized messages should be provided whenever possible, using the msgDataElement element of the Common Base Event.

msgLocale

The message locale should be provided 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 it should be provided within problem

determination events whenever it is available. The only reason this is not a required field for problem determination events is there are instances where the locale information is not provided or available when formatting the Common Base Event.

msgId and msgIdType
Several companies include within internationalized message text a locale-independent identifier that can be used to interpret the condition being described by the message text, independent of the language 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 the location (and language) where it was issued, meaning it 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. These fields should not be supplied when the message text has not been 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 it should be provided whenever the Common Base Event includes localized or translated message text, for example when providing problem determination events representing issued messages or log events. It is not a required field for problem determination events because not all problem determination events include translated message text, and there are cases where the value is not provided or available when formatting the Common Base Event. This field should not be supplied when the message text has not been 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 it should be provided whenever the Common Base Event includes localized or translated message text, for example when providing problem determination events representing issued messages or log events. It is not a required field for problem determination events because not all problem determination events include translated message text, and there are cases where the value is not provided or available when formatting the Common Base Event. This value contains the list of locale independent values (message tokens) inserted into the localized message text when creating a translated message. It is very difficult to extract these values from a translated message without having knowledge of the translated message template used to create the message, meaning having these values separate from the message text is invaluable for locale-independent and programmatic analysis of the data supplied in the message text. This field should not be supplied when the message text has not been translated or localized, e.g. for trace events. Note: 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. The msgCatalogTokens field should always be used to supply the list of message tokens included in the message text associated with an event. These values can also be supplied in other parts of the Common Base Event as well, 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 they should be provided whenever the Common Base Event includes localized or translated message text, for example when providing problem determination events representing issued messages or log events. They are not required fields for problem determination events because not all problem determination events include translated message text, and there are cases where the values are not provided or available when formatting the Common Base Event. These fields should not be supplied when the message text has not been translated or localized, for example, for trace events.

Extended Data

The base information included in a Common Base Event may not be sufficient to represent all of the information captured by a component when creating a problem determination event. 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. We recommend using the latter approach, ExtendedDataElement elements.

An ExtendedDataElement element is used to represent a single data item, and 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 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

The XML document shown below is an example of a Common Base Event (CBE) instance generated by a WebSphere Application Server application.

```
<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>
```

Note that there are a number of extendedDataElement elements in the actual XML which are used by WebSphere Application Server, but which are not for use by applications as they may change without notice in future releases.

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

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

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

The attributes of both the sourceComponentId and the reporterComponentId elements are the same. They identify the component's type, name, operating system and network location. The content of these attributes provides vertical correlation of the stack of IT resources active when the when the CBE was created.

Also included in the CommonBaseEvent element are contextDataElements that describe the context in which the situation occurred. This context correlates CBE instances that were part of the same piece of work. This is called horizontal correlation since an instance of a particular context type correlates events at the same level of abstraction (for example at the business level, or at the application level, or at the middleware level.)

Finally, there are the extended data elements. These contain additional data used to describe to situation. In this example, there is an extended data element added by WebSphere Application Server to describe the J2EE component that generated the CBE instance and some application data.

Sample Common Base Event template

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. Template information is used by the content handler to fill in blanks in the Common Base Event when the Common Base Event complete method is called. Information that is already supplied in the event will not be overridden if the same field is supplied in the template.

The following is an example of 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. The following definitions and examples should be used when creating Common Base Events for problem determination.

Business Application

A business application is the business logic and business data 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 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, developed either by the enterprise or a vendor, managed by the enterprise. A components may be created by the enterprise, typically for usage within a specific business application. For example, the ACME Corporation may create a set of enterprise beans to represent the business logic required by their Payroll Application. A component may also be an asset 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, may 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 may 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

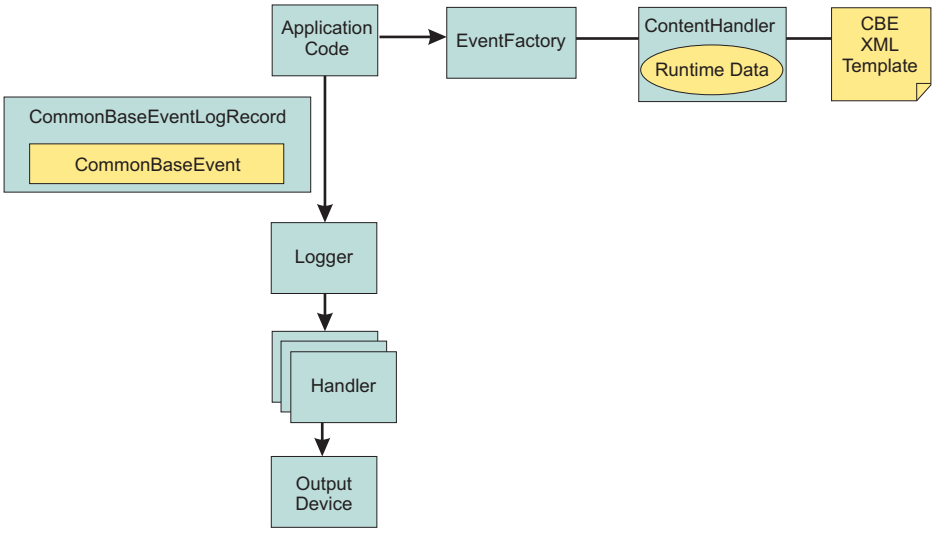
This topic describes how WebSphere Application Server takes advantage of the Common Base Events.

WebSphere Application Server uses Common Base Events within its logging framework. Common Base Events can be created explicitly and then logged via 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.

Using the Common Base Event API with the Java Logging API to log Common Base Events

In cases where the events generated by the Java logging API are insufficient to describe the event that needs to be captured, Common Base Events can be created using 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 CommonBaseEvents:



The steps for generating a Common Base Event are as follows:

1. Application code invokes createCommonBaseEvent method on EventFactory to create a CommonBaseEvent.
2. Application code wraps CommonBaseEvent in a CommonBaseEventLogRecord, and adds event specific data.
3. Application code calls CommonBaseEvent's complete() method.
4. CommonBaseEvent invokes ContentHandler's completeEvent() method.
5. ContentHandler adds XML template data to CommonBaseEvent (including for example, the component name). Note that not all ContentHandlers support templates.
6. ContentHandler adds runtime data to CommonBaseEvent (including for example, the current thread name).
7. Application code passes CommonBaseEventLogRecord to Logger using Logger.log method.
8. Logger passes CommonBaseEventLogRecord to Handlers.
9. Handlers format data and write to output device.

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 ensures consistency of Common Base Event content across events. However, other Common Base Event factories can be create and used. See "Configuring Common Base Events for an application" on page 98 for details on how to create and use custom event factories.

Common Base Event factory context:

The event factory context provides a service to look up event factory homes. The event factory context can be retrieved using a call to EventFactoryContext.getInstance(). 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 must be located on the classpath to be found. The EventFactoryContext also stores an EventFactoryHome as a default, which can be obtained with a call to EventFactoryContext.getInstance().getEventFactoryHome().

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

More details can be found in the javadoc for org.eclipse.hyades.logging.events.cbe.EventFactory.

Common Base Event factory home:

Event Factory homes provide Event Factory instantiation based on a unique factory name. Event Factory home implementations are tightly coupled with content handlers which 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 based on the type of the Event Factory home implementation.

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

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

Common Base Event factory:

Event factories allow you to create Common Base Events and complete event properties using 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 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 only be retrieved from their associated event factory home. Event factory instances are retrieved and maintained based on unique names. Event factory names are hierarchal; they are represented using the standard Java dot-delimited name-space naming conventions.

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

Common Base Event content handler:

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

The use of content handlers is recommended to ensure consistency of field usage in the Common Base Event within a component or within a set of components sharing the same runtime. For example, some content handlers allow a template to be specified. If used consistently across a component, this would ensure that all events for that component would have the same template info filled in. Similarly, some content handlers can also supply runtime information to their associated Common Base Events. If consistently used throughout the entire runtime, this would ensure that all events use runtime data in a similar fashion.

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

More details can be found in the javadoc for org.eclipse.hyades.logging.events.cbe.ContentHandler at www.eclipse.org/hyades.

Using the Java Logging API to Generate and Log Common Base Events

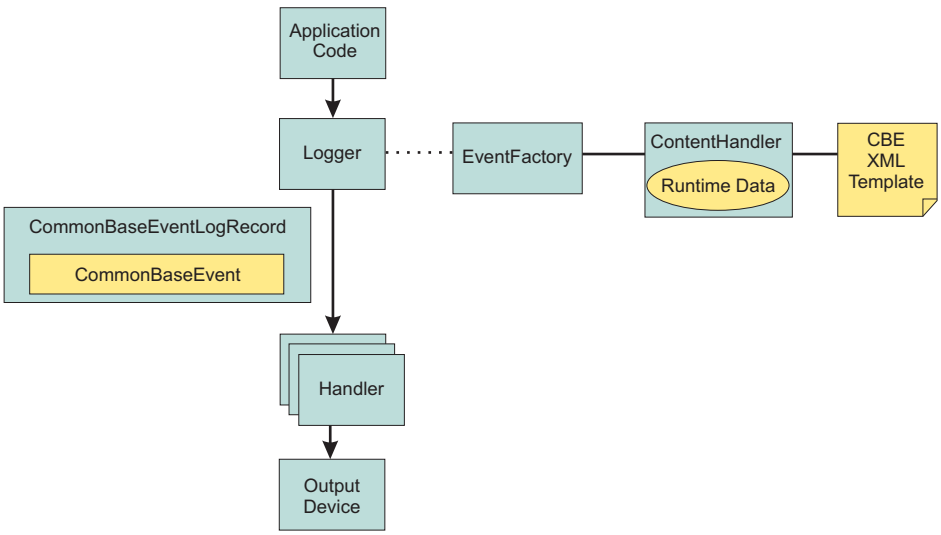
In the WebSphere Application Server, the Java logging API (java.util.logging) automatically creates Common Base Events for events logged at level WsLevel.DETAIL or above (including WsLevel.DETAIL, Level.CONFIG, Level.INFO, WsLevel.AUDIT, Level.WARNING, Level.SEVERE, and WsLevel.FATAL). These Common Base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

The java.util.logging.Logger class provides a variety of methods with which data can be logged. The WebSphere Application Server uses a special implementation of the 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 WsLevel.DETAIL or more severe levels
- logp - when used with WsLevel.DETAIL or more severe levels
- logrb - when used with WsLevel.DETAIL or more severe levels

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

The following diagram illustrates how application code can create log CommonBaseEvents:



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

1. Application code invokes named Logger (WsLevel.DETAIL or above) with event specific data.
2. Logger creates a CommonBaseEvent using createCommonBaseEvent method on EventFactory associated to the Logger (see Configuring Common Base Events for an application).

3. Logger creates a CommonBaseEvent using EventFactory associated to the Logger
4. Logger wraps CommonBaseEvent in a CommonBaseEventLogRecord, and adds event specific data.
5. Logger calls CommonBaseEvent's complete() method.
6. CommonBaseEvent invokes ContentHandler's completeEvent() method.
7. ContentHandler adds XML template data to CommonBaseEvent (including for example, the component name). Note that not all ContentHandlers support templates.
8. ContentHandler adds runtime data to CommonBaseEvent (including for example, the current thread name).
9. Logger passes CommonBaseEventLogRecord to Handlers.
10. Handlers format data and write to Output Device.

Configuring Common Base Events for an application

The logger.properties file allows you to set logger attributes for your component. The properties file is loaded the first time the method `Logger.getLogger(`*`loggername`*`)` is called within an application. The logger.properties file must be either on the WebSphere Application Server class path, or the context class path. See "Configuring logging properties for an application" on page 71 for details on the logger.properties file

To use the Common Base Events standard for events generated by your application, include the eventfactory property in your logger.properties file. The event factory property specifies the name of the Common Base Event template to use with the event factory. See "Sample Common Base Event template" on page 93 for details on the Common Base Event template. Note that the naming convention for this is the fully qualified component name with the extension ".event.xml". For example, a template that applies to package com.ibm.compXYZ would be called "com.ibm.compXYZ.event.xml"

Sample logger.properties file

In the following sample, event factory com.ibm.xyz.MyEventFactory will be used by any loggers in the com.ibm.websphere.abc package or any sub-packages which do not have their own configuration file.

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

Common Base Event content generated when using the WebSphere Application Server default event factory

When no other event factory is configured for a logger, the WebSphere Application Server uses its default event factory for creation of Common Base Events. The content handler associated with the default event factory populates fields as follows:

Field	Value	Notes
CommonBaseEvent.globalInstanceId	unique record id	only set if CommonBaseEvent.globalInstanceId was null before completeEvent() was called
CommonBaseEvent.msg	localized message based on MsgDataElement	only set if CommonBaseEvent.msg was null before completeEvent() was called
CommonBaseEvent.severity	set based on value of Level set on CommonBaseEventLogRecord if level >= Level.SEVERE set to 50 else if level >= Level.WARNING set to 30 default set to 10	only set if CommonBaseEvent.severity was null before completeEvent() was called

Field	Value	Notes
CommonBaseEvent.Component-Identification.component	set based on value of LoggerName set on CommonBaseEventLogRecord	only set if CommonBaseEvent.Component-Identification.component was null before completeEvent() was called
CommonBaseEvent.Component-Identification.componentIdType	″Unknown″	only set if CommonBaseEvent.Component-Identification.componentIdType was null before completeEvent() was called
CommonBaseEvent.Component-Identification.executionEnvironment	OSname[OSarch]#OSversion	only set if CommonBaseEvent.Component-Identification.executionEnvironment was null before completeEvent() was called
CommonBaseEvent.Component-Identification.instanceId	cellName\nodeName\serverName	only set if CommonBaseEvent.Component-Identification.instanceId was null before completeEvent() was called only set in a server environment (ignored in a client application)
CommonBaseEvent.Component-Identification.location	hostname	only set if both CommonBaseEvent.Component-Identification.location and CommonBaseEvent.Component-Identification.locationType were null before completeEvent() was called
CommonBaseEvent.Component-Identification.locationType	″Hostname″	only set if both CommonBaseEvent.Component-Identification.location and CommonBaseEvent.Component-Identification.locationType were null before completeEvent() was called
CommonBaseEvent.Component-Identification.processId	internally generated representation of process number	only set if CommonBaseEvent.Component-Identification.processId was null before completeEvent() was called
CommonBaseEvent.Component-Identification.subComponent	set based on values of sourceClassName and sourceMethodName set on CommonBaseEventLogRecord sourceClassName.sourceMethodName	CommonBaseEvent.Component-Identification.subComponent was null before completeEvent() was called and both sourceClassName and sourceMethodName were set
CommonBaseEvent.Component-Identification.threadId	set to value of JVM thread name	only set if CommonBaseEvent.Component-Identification.threadId was null before completeEvent() was called
CommonBaseEvent.Component-Identification.componentType	″http://www.ibm.com/namespaces/-autonomic/WebSphereApplication-Server″	only set if CommonBaseEvent.Component-Identification.componentType was null before completeEvent() was called
CommonBaseEvent.MsgData-Element.msgLocale	set based on default locale of JVM	only set if CommonBaseEvent.msg was null before completeEvent() was called

Field	Value	Notes
CommonBaseEvent.Situation. -categoryName	″ReportSituation″	only set if CommonBaseEvent.Situation was null before completeEvent() was called
CommonBaseEvent.Situation. -situationType.type	″ReportSituation″	only set if CommonBaseEvent.Situation was null before completeEvent() was called
CommonBaseEvent.Situation. -situationType.reasoningScope	″EXTERNAL″	only set if CommonBaseEvent.Situation was null before completeEvent() was called
CommonBaseEvent.Situation. -situationType.reportCategory	″LOG″	only set if CommonBaseEvent.Situation was null before completeEvent() was called

Note that the sourceComponentIdentification is populated if no reporterComponentIdentification exists when completeEvent is invoked on the ContentHandler, otherwise the reporterComponentIdentification will be populated instead.

The same content handler that WebSphere Application Server uses for generating its events can be used in your applications. Event factory instances can be obtained as follows:

```
EventFactory eventFactory = EventFactoryContext.getInstance().getEventFactoryHome().getEventFactory(factoryName);
```

where factoryName is the name of the CommonBaseEvent template you wish to use with the factory.

The factoryName can also be specified as a configuration parameter for a logger. See “Configuring Common Base Events for an application” on page 98 for more details.

Best Practices for Logging Common Base Events in WebSphere Application Server

The following practices will ensure consistent use of Common Base Events within your components and between your components and WebSphere Application Server components:

- Use a different Logger for each component. Sharing loggers across components gets in the way of being able to associate Loggers with component specific information
- Associate Loggers with event templates that specify source component identification. This 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 CommonBaseEvents until you are finished adding data to the CommonBaseEvent and are ready to log it. This ensures that any decisions made by the ContentHandler based on data already in the event will be made using the final data.

The following sample logger.properties file entry demonstrates how to associate Logger com.ibm.componentX with event factory com.ibm.componentX:

```
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>
```

Programming with the JRas framework

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

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

Understanding the JRas facility

Note: The JRas framework 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 has failed and give information as to why. This enables the administrator to take the proper corrective action. Those who develop or maintain applications might need to gather detailed information relating to the execution path of a running application in order to determine the root cause of a failure that is due to a code bug. The facilities that are used for these purposes are typically referred to as message logging and diagnostic trace.

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

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

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

WebSphere Application Server provides a message logging and diagnostic trace API that can be used by applications. This API is based on the 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 (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 required for integration into the WebSphere Application Server runtime or for usage in a J2EE environment. To overcome these limitations, WebSphere Application Server provides a set of extension classes to address these shortcomings. This collection of extension classes is referred to as the JRas extensions. The JRas extensions do not modify the interfaces introduced by the stand-alone JRas logging toolkit, but simply provide the appropriate implementation classes. The conceptual structure introduced by the stand-alone JRas logging toolkit is described below. It is equally applicable to the JRas extensions.

JRas Concepts

The following is a basic overview of important concepts and constructs introduced by the stand-alone JRas logging toolkit. It is not meant to be an exhaustive overview of the capabilities of this logging toolkit, nor is it intended to be a detailed discussion of usage or programming paradigms. More detailed information, including code examples, is available in JRas extensions and its subtopics, including in the javadoc for the various interfaces and classes that make up the logging toolkit.

Event Types

The 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. These are message loggers and trace loggers. The set of methods on message loggers and trace loggers are different, since they provide different functionality. Message loggers create only message records and trace loggers create only trace records. Both types of loggers contain masks that indicates which categories of events the logger should process and which it should ignore. Although every JRas logger is defined to contain both a message and trace mask, the message logger only uses the message mask and the trace logger only uses the trace mask. For example, by setting a message logger's message mask to the appropriate state, it can be configured to process only Error messages and ignore Informational and Warning messages. Changing the state of a message logger's trace mask has no effect.

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

Methods that are used within the logging infrastructure itself should not make calls to the logger method. When an application uses an object that extends a thread class, implements the hashCode(), 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 will process. For example, a message logger may be configured to pass both warning and error events, but a handler attached to the

message logger may be configured to only pass error events. Handlers also include formatters, which the handler invokes to format the data in the passed event before it is written to the output device.

Formatters

Handlers are configured with formatters, which know how to format events of certain types. A handler may contain multiple formatters, each of which 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

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

JRas extensions

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

Usage Model

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

Integrated

In this mode, message and trace records are written only to logs defined and maintained by the WebSphere Application Server runtime. This is the default mode of operation and is equivalent to the WebSphere Application Server 4.0 mode of operation.

stand-alone

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

Combined

In this mode message and trace records are written to both WebSphere Application Server runtime logs and to 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 there 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 has been defined that allow the JRas extensions to be used in either a stand-alone or combined mode of operations.

JRas extension classes

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

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

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

WebSphere Application Server Message and Trace loggers

The message and trace loggers provided by the 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, described below. You cannot directly instantiate message and trace loggers. Obtaining loggers in any manner other than directly from the Manager is not allowed. Doing so is a direct violation of the programming model.

The message and trace loggers instances obtained from the WebSphere Application Server Manager class are subclasses of the `RASMessageLogger()` and `RASTraceLogger()` classes provided by the stand-alone JRas logging toolkit. The `RASMessageLogger()` and `RASTraceLogger()` classes define the set of methods that are directly available. Public methods introduced by the JRas extensions logger subclasses cannot be called directly by user code. Doing so is a violation of the programming model.

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

In general, loggers have no predefined granularity or scope. A single logger could be used to instrument an entire application. Or users may determine that having a logger per class is more desirable. Or the appropriate granularity may lie somewhere in between. Partitioning an application into logging domains is rightfully determined by the application writer.

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

Please note that there is no provision for allowing users to provide their own logger subclasses for use in a WebSphere Application Server environment.

WebSphere Application Server handlers

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

WebSphere Application Server formatters

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

WebSphere Application Server manager

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

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

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

Extending the JRas framework

Since the Jras extensions classes do not provide the flexibility and behavior required for many scenarios, a variety of extension points have been defined. You are allowed to write your own implementation classes to obtain the required behavior.

Note: The JRas framework described in this task and its sub-tasks 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 to allow you to provide your own message or trace logger subclasses. In general, user-provided extensions cannot be used to affect the integrated mode of operation. The behavior of the integrated mode of operation is solely determined by the WebSphere Application Server 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. You should directly create instances of user-defined handlers and add them to the loggers obtained from the Manager.

The stand-alone JRas logging toolkit provides several handler implementation classes. These handler classes are inappropriate for usage in the J2EE environment. You cannot directly use or subclass any of the Handler classes provided by the 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 only add these classes to a user-defined handler. WebSphere Application Server handlers cannot be configured to use user-defined formatters. Instead, directly create instances of your formatters and add them to the your handlers appropriately.

As with handlers, the stand-alone JRas logging toolkit provides several formatter implementation classes. Direct usage 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 retrieved from the Manager have their message masks set to *pass* or process all message event types defined in the `RASIMessageEvent` interface. In order to process user-defined message types, you must manually set the message logger mask to the appropriate state by user code after the message logger has been obtained from the Manager. WebSphere Application Server does not provide any built-in systems management support for managing any message types.

Message event objects

The 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 currently registered with that logger.

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

Trace event types

The 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 defined in the `RASITraceEvent` interface.

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

In order to process user-defined trace types, the trace logger mask must be manually set to the appropriate state by user code. WebSphere Application Server systems management facilities cannot be used to manage user-defined trace types, either at start time or 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 handlers currently registered with that logger.

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

User defined types, user defined events and WebSphere Application Server

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

Writing User Extensions:

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. Since user written extensions typically access such protected resources, user written extensions must contain the appropriate security checking calls, using AccessController `doPrivileged()` calls. In addition, the user written extensions must contain the appropriate policy file. In general, it is recommended that you locate user written extensions in a separate package. It is your responsibility to restrict access to the user written extensions appropriately.

Writing a handler

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

RASIObject interface

The `RASIObject` interface is the base interface for 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 methods*`public Hashtable getConfig()`* and *`public void setConfig(Hashtable ht)`* are used to get and set the configuration state. The JRas extensions do not support properties based configuration and it is recommended that these methods be implemented as no-operations. You can implement your own properties based configuration using these methods.
- Loggers, handlers and formatters can be named objects. For example, the JRas extensions require the user to provide a name for the loggers that are retrieved from the manager. You can name your handlers. The methods *`public String getName()`* and *`public void setName(String name)`* are provided to get or set the name field. The JRas extensions currently do not call these methods on user handlers. You can implement these methods as you want, including as no operations.
- Loggers, handlers and formatters can also contain a description field. The methods *`public String getDescription()`* and *`public void setDescription(String desc)`* can be used to get or set the description field. The JRas extensions currently do not use the description field. You can implement these methods as you want, including as no operations.

- The method *public String getGroup()* is provided for usage by the RASManager. Since the JRas extensions provide their own Manager class, this method is never called. It is recommended you implement this as a no-operation.

RASIMaskChangeGenerator interface

The RASIMaskChangeGenerator interface is the interface that defines the implementation methods for filtering of events based on a mask state. This means that it is currently implemented by both loggers and handlers. By definition, an object that implements this interface contains both a message mask and a trace mask, although both need not be used. For example, message loggers contain a trace mask, but the trace mask is never used since the message logger never generates trace events. Handlers however can actively use both mask values. For example a single handler could handle both message and trace events.

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

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

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

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

This means that all handlers are required to properly implement these methods. In addition, when a user handler is instantiated, the logger it is to be added to should be registered with the handler using the *addMaskChangeListener()* method. When either the message mask or trace mask of the handler is changed, the logger must be called back to inform it of the mask change. This allows the logger to dynamically maintain the composite mask.

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

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

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

RASIHandler interface

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

The `RASIHandler` interface as provided by the 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. Since spawning of threads is not allowed in the WebSphere Application Server environment, it is expected that handlers will not queue or batch events, although this is not expressly prohibited.

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

Writing a formatter

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

RASIFormatter interface

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

Programming model summary

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

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

As described previously, you can use the WebSphere Application Server JRas extensions in three distinct operational modes. The programming models concepts and restrictions apply equally across all modes of operation.

- You must not use implementation classes provided by the stand-alone JRas logging toolkit directly, unless specifically noted otherwise. Direct usage of those classes is not supported. IBM Support will provide no diagnostic aid or bug fixes relating to direct usage of classes provided by the stand-alone JRas logging toolkit.
- You must obtain message and trace loggers directly from the Manager class. You cannot directly instantiate loggers.
- There is no provision that allows you to replace the WebSphere Application Server message and trace logger classes.
- You must guarantee that the logger names passed to the Manager are unique, and follow the naming constraints documented below. Once a logger is obtained from the Manager, you must not attempt to change the name of the logger by calling the `setName()` method.
- Named loggers can be used more than once. For any given name, the first call to the Manager results in the Manager creating a logger that is associated with that name. Subsequent calls to the Manager that specify the same name result in a reference to the existing logger being returned.
- The Manager maintains a hierarchical namespace for loggers. It is recommended but not required that a dot-separated, fully qualified class name be used to identify any given logger. Other than dots or periods, logger names cannot contain any punctuation characters, such as asterisk (*), comma(.), equals sign(=), colon(:), or quotes.
- Group names must comply with the same naming restrictions as logger names.
- The loggers returned from the Manager are subclasses of the RASMessageLogger and RASTraceLogger provided by the stand-alone JRas logging toolkit. You are allowed to call any public method defined by the RASMessageLogger and RASTraceLogger classes. You are not allowed to call any public method introduced by the provided subclasses.
- If you want to operate in either stand-alone or combined mode, you must provide your own Handler and Formatter subclasses. You are not allowed to use the Handler and Formatter classes provided by the stand-alone JRas logging toolkit. User written handlers and formatters must conform to the documented guidelines.
- Loggers obtained from the manager come with a WebSphere Application Server handler installed. This handler will write message and trace records to logs defined by the WebSphere Application Server runtime. Manage these logs using the provided systems management interfaces.
- You can programmatically add and remove user-defined handlers from a logger at any time. Multiple additions and removals of user defined handlers are allowed. You are responsible for creating an instance of the handler to add, configuring the handler by setting the handler's mask value and formatter appropriately, then adding the handler to the logger using the `addHandler()` method. You are responsible for programmatically updating the masks of user-defined handlers as appropriate.
- You may get a reference to the handler installed within a logger by calling the `getHandlers()` method on the logger and processing the results. You must not call any methods on the handler obtained in this fashion. You are allowed to remove the WebSphere Application Server handler from the logger by calling the logger's `removeHandler()` method, passing in the reference to the WebSphere Application Server handler. Once removed, the WebSphere Application Server handler cannot be re-added to the logger.
- You are allowed to define your own message type. The behavior of user-defined message types and restrictions on their definitions is discussed in Extending the JRas framework.
- You are allowed to define your own message event classes. The usage of user-defined message event classes is discussed in Extending the JRas framework.
- You are allowed to define your own trace types. The behavior of user-defined trace types and restrictions on your definitions is discussed in Extending the JRas framework.
- You are allowed to define your own trace event classes. The usage of user-defined trace event classes is discussed in Extending the JRas framework.
- You must programmatically maintain the bits in the message and trace logger masks that correspond to any user-defined types. If WebSphere Application Server facilities are being used to manage the predefined types, these updates must not modify the state of any of the bits corresponding to those types. If you are assuming ownership responsibility for the predefined types then you can change all bits of the masks.

JRas Messages and Trace event types

This section describes JRas message and trace event types.

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

Event types

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

Platform Message Event Types

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

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

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

Mapping WebSphere Application Server trace types to JRas `RASITraceEvent` ″Level″ types.

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

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

WebSphere Application Server native type	JRas RASITraceEvent enumerated types
`Event`	TYPE_ERROR_EXC, TYPE_SVC, TYPE_OBJ_CREATE, TYPE_OBJ_DELETE

EntryExit	TYPE_ENTRY_EXIT, TYPE_API, TYPE_CALLBACK, TYPE_PRIVATE, TYPE_PUBLIC, TYPE_STATIC
Debug	TYPE_MISC_DATA

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

Message and Trace parameters

The various message logging and trace method signatures accept parameter types of `Object`, `Object[]` and `Throwable`. WebSphere Application Server will process and format the various parameter types as follows.

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

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

`Object[]`
The `Object[]` is provided for the case when more than one parameter is passed to a message logging or trace method. `toString()` is called on each `Object` in the array. Nested arrays are not handled. (i.e. none of the elements in the `Object` array should be an array).

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

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

- [B@924586a0b - This is deciphered as ″a byte array at location X″. This is typically returned when an array is passed as a member of an `Object[]`. It is the result of calling `toString()` on the `byte[]`.
- Illegal trace argument : array of long. This is typically returned when an array of primitives is passed to a method taking an `Object`.
- 01040703... : the hex representation of an array of bytes. Typically this may be seen when a byte array is passed to a method taking a single `Object`. This behavior is subject to change and should not be relied on.
- ″1″ ″2″ ... : The String representation of the members of an `int[]` formed by converting each element to an Integer and calling `toString` on the Integers. This behavior is subject to change and should not be relied on.
- [Ljava.lang.Object;@9136fa0b : An array of objects. Typically this is seen when an array containing nested arrays is passed.

Controlling message logging

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

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

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

Some editions of the product allow the user to specify a message filter level for a server process. When such a filter level is set, only messages at the specified severity levels are written to WebSphere Application Server logs. This means that messages types that pass the message logger's mask check may be filtered out by the WebSphere Application Server itself.

Controlling Tracing

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

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

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

When the dynamic trace enablement feature available on some platforms is used, the trace state change is reflected both in the Application Server run-time and the trace loggers trace mask. If user code programmatically changes the bits in the trace mask corresponding to the values defined by in the `RASITraceEvent` interface, the trace logger's mask state and the run-time state will become unsynchronized and unexpected results will occur. Therefore, programmatically changing the bits of the mask corresponding to the values defined in the `RASITraceEvent` interface is not allowed.

Instrumenting an application with JRas extensions

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

To instrument an application using the WebSphere Application Server JRas extensions, perform the following steps:
1. Determine the mode the extensions will be used in: integrated, stand-alone or combined.
2. If the extensions will be used in either stand-alone or combined mode, create the necessary handler and formatter classes.
3. If localized messages will be used by the application, create a resource bundle as described in Creating JRas resource bundles and message files.

4. In the application code, get a reference to the Manager class and create the manager and logger instances as described in Creating JRas manager and logger instances.
5. Insert the appropriate message and trace logging statements in the application as described in Creating JRas manager and logger instances.

Creating JRas resource bundles and message files

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

The WebSphere Application Server message logger provides the `message()` and `msg()` methods to allow the user to log localized messages. In addition, it provides the `textMessage()` method for logging of messages that are not localized. Applications can use either or both, as appropriate.

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

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

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

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

The two techniques are described as follows:

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

Late binding
: The application can choose to have the WebSphere Application Server runtime localize the message in the process where it is displayed. Using this technique,the resource bundles are packaged in a 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 administrator's console or log viewing program might be run. You must install the `.jar` file in a directory that is part of the extensions classpath. In addition, if you forward logs to IBM service, you must also forward the `.jar` file containing the resource bundles.

To create a resource bundle, perform the following steps.

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 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 only of white space, denotes a single property. A backslash (\) is treated as the line-continuation character.
- The syntax for a property file consists of a key, a separator, and an element. Valid separators include the equal sign (=), colon (:), and white space ().
- The key consists of all characters on the line from the first non-white space character to the first separator. Separator characters can be included in the key by escaping them with a backslash (\), but doing this is not recommended, because escaping characters is error prone and confusing. It is instead recommended that you use a valid separator character that does not appear in any keys in the properties file.
- White space after the key and separator is ignored until the first non-white space character is encountered. All characters remaining before the line-termination character define the element.

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

2. The file can then be translated into localized versions of the file with language-specific file names (for example, a file named `DefaultMessages.properties` can be translated into DefaultMessages_de.properties for German and DefaultMessages_ja.properties for Japanese).
3. When the translated resource bundles are available, write them to a system-managed persistent storage medium. Resource bundles are then used to convert the messages into the requested national language and locale.
4. When a message logger is obtained from the JRas manager, it can be configured to use a particular resource bundle. Messages logged via the `message()` API will use this resource bundle when message localization is performed. At run time, the user's locale setting is used to determine the properties file from which to extract the message specified by a message key, thus ensuring that the message is delivered in the correct language.
5. 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's location relative to any directory in the classpath. For instance, if the property resource bundle named DefaultMessages.properties is located in the *baseDir*/*subDir1*/*subDir2*/`resources` directory and *baseDir* is in the class path, the name *subdir1*.*subdir2*.`resources.DefaultMessage` is passed to the message logger to identify the resource bundle.

Developing JRas resource bundles:

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

Resource bundle sample

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

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

Assume that the base directory for the application that uses this resource bundle is `″baseDir″` and that this directory will be in the classpath. Assume that the properties file is stored in a subdirectory of `baseDir` that is not in the classpath (e.g. `baseDir/subDir1/subDir2/resources`). In order to allow the messages file to

be resolved, the name subDir1.subDir2.resources.DefaultMessage is used to identify the PropertyResourceBundle and is passed to the message logger.

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

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

Once the file DefaultMessages.properties is created, the file can be sent to a translation center where the localized versions will be generated.

Creating JRas manager and logger instances

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

Note: 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. See Setting up for integrated JRas operation for information on configuring for this mode of operation.

In the combined mode, message and trace events are logged to both WebSphere Application Server and user-defined logs. See Setting up for combined JRas operation for more information on configuring for this mode of operation.

In the stand-alone mode, message and trace events are logged only to user-defined logs. See Setting up for stand-alone JRas operation for more information on configuring for this mode of operation.

Using the message and trace loggers

Regardless of the mode of operation, the use of message and trace loggers is the same. See Creating JRas resource bundles and message files for more information on using message and trace loggers.

Using a message logger

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

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

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

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

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

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

Using a trace logger

Since trace is normally disabled, trace methods should be guarded for performance reasons.

```
private void methodX(int x, String y, Foo z)
{
   // trace an entry point. Use the guard to make sure tracing is enabled.
Do this checking before we waste cycles gathering parameters to be traced.
   if (trcLogger.isLoggable(RASITraceEvent.TYPE_ENTRY_EXIT) {
        // since I want to trace 3 parameters, package them up in an Object[]
        Object[] parms = {new Integer(x), y, z};
        trcLogger.entry(RASITraceEvent.TYPE_ENTRY_EXIT, this, "methodX", parms);
  }
... logic
  // a debug or verbose trace point
  if (trcLogger.isLoggable(RASITraceEvent.TYPE_MISC_DATA) {
        trcLogger.trace(RASITraceEvent.TYPE_MISC_DATA, this, "methodX" "reached here");
  }
  ...
  // Another classification of trace event. Here an important state change
has been detected, so a different trace type is used.
  if (trcLogger.isLoggable(RASITraceEvent.TYPE_SVC) {
     trcLogger.trace(RASITraceEvent.TYPE_SVC, this, "methodX", "an important event");
  }
 ...
  // ready to exit method, trace. No return value to trace
    if (trcLogger.isLoggable(RASITraceEvent.TYPE_ENTRY_EXIT)) {
        trcLogger.exit(RASITraceEvent.TYPE_ENTRY_EXIT, this, "methodX");
   }
}
```

Setting up for integrated JRas operation

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 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 and create the loggers. Since loggers are named singletons, you can do this in a variety of places. One logical candidate for enterprise beans is the `ejbCreate()` method. For example, for the enterprise bean named ″myTestBean″, place the following code in the `ejbCreate()` method.

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



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

Setting up for combined JRas operation

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 have written a user defined handler named `SimpleFileHandler` and a user defined formatter named `SimpleFormatter`. It also assumes that you are not using user defined types or events.

1. Import the requisite JRas extensions classes

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

2. Import the user handler and formatter.

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

3. Declare the logger references.

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

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

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

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

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

Setting up for stand-alone JRas operation

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`. It is also assumes that no user-defined types or events are being used.

1. Import the requisite JRas extensions classes

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

2. Import the user handler and formatter.

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

3. Declare the logger references.

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

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

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

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

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

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

Chapter 5. 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 investigating common problems organized according to tasks within WebSphere Application server, see Troubleshooting by task.
2. For tips on how to investigate common kinds of problems based on the component that is causing the problem, see Troubleshooting by component.
3. If you already have an error message and want to quickly look up its explanation and recommended response, look up the message by selecting the **Reference** view of the information center navigation and expanding **Messages**.
4. For help in knowing where to find error and warning messages, interpreting messages, and configuring log files, see Working with message logs.
5. Difficult problems can require the use of tracing, which exposes the low-level flow of control and interactions between components. For help in understanding and using traces, see Working with trace.
6. For help in adding log and trace capability to your own application, see “Logging and tracing with Java logging” on page 63.
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 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.
9. The IBM Developer Kit and Runtime Environment, Java 2 Technology Edition, Version 1.4.1 Diagnostics Guide 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. You can find the guide at http://www.ibm.com/developerworks/java/jdk/diagnosis/.
10. For current information available from IBM Support on known problems and their resolution, see the IBM Support page.
11. 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.

Message reference

You can log WebSphere Application Server system messages from a variety of sources, including application server components and applications. Messages logged by application server components and associated IBM products start with a unique message identifier that indicates the component or application that issued the message. The message identifier can be either 8 or 9 characters in length and has the form:

```
CCCC1234X
```

where:

CCCC is a four character alphabetic component or application identifier.
1234 is a 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 the topic **Messages**.

CORBA minor codes

Overview

Common Object Request Broker Architecture (CORBA) is an industry-wide standard for object-oriented communication between processes, which is supported in several programming languages. Several subcomponents of WebSphere Application Server use CORBA to communicate across processes.

When a CORBA process fails, that is a request from one process to another cannot be sent, completed, or returned, a high-level exception is created, such as TransactionRolledBackException: CORBA TRANSACTION_ROLLEDBACK. To show the underlying cause of the failure, applications that use CORBA services generate minor codes that are written to the exception stack. Look for ″minor code″ in the exception stack to locate these exceptions.

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 240
0x49421050-0x4942105F, 0x49421070-0x4942107F	ORB services	“Object request broker component troubleshooting tips” on page 43
0x4f4d and above	Standard CORBA exceptions	http://www.omg.org
0x49421080-0x4942108F	Naming services	See Reference: Generated API documentation for information about the `ws.code.naming.src.com.ibm.websphere.naming.WsnCorbaMinorCodes` class.
0x49421080-0x4942108F	Workload Management	See Reference: Generated API documentation for information about the `com.ibm.websphere.wlm.WsCorbaMinorCodes` class.

Working with message logs

WebSphere Application Server can write system messages to several general purpose logs. These include the JVM logs, the process logs and the IBM service log.

The JVM logs are created by redirecting the `System.out` and `System.err` streams of the JVM to independent log files. WebSphere Application Server writes formatted messages to the `System.out` stream. In addition, applications and other code can write to these streams using the `print()` and `println()` methods defined by the streams. Some 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 Analyzer instead. The `System.err` log contains exception stack trace information that is useful when performing problem analysis.

Since each application server represents a JVM, there is one set of JVM logs for each application server and all of its applications, located by default in the *installation_root*/profiles/*profile_name*/logs/*server_name* directory. In the case of a WebSphere Application Server Network Deployment configuration, JVM logs are also created for the deployment manager and each node manager, since they also represent JVMs.

The process logs are created by redirecting the stdout and stderr streams of the process to independent log files. Native code, including the Java virtual machine (JVM) itself 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 manager.

The IBM service log contains both the WebSphere Application Server messages that are written to the System.out stream and some special messages that contain extended service information that is normally not of interest, but can be important when analyzing problems. There is one service log for all WebSphere Application Server JVMs on a node, including all application servers. The IBM Service log is maintained in a binary format and requires a special tool to view. This viewer, the Log Analyzer, provides additional diagnostic capabilities. In addition, the binary format provides capabilities that are utilized by IBM support organizations.

In addition to these general purpose logs, WebSphere Application Server contains other specialized logs that are 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 HTTP server and Application Server are working separately, but requests are not passing from HTTP server to Application Server.

Viewing the JVM logs

The JVM logs are written as plain text files. Therefore there are no special requirements to view these logs. They are located in the *installation_directory*/profiles/*profile_name*/logs/*server_name* directory, and by default are named SystemOut.log and SystemErr.log.

There are two techniques that you can use to view the JVM logs for an application server:
- Use the adminstrative console. This supports viewing the JVM logs from a remote machine.
- Use a text editor on the machine where the logs are stored.

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. Open the file in a text editor or drag and drop the file into an editing and viewing program.

Interpreting the JVM logs

The JVM logs contain print data written by applications. The application can write this data written directly in the form of System.out.print(), System.err.print(), or other method calls. The application can also

write data indirectly by calling a JVM function, such as an `Exception.printStackTrace()`. In addition, the `System.out` JVM log contains system messages written by the WebSphere Application Server.

You can format application data to look like WebSphere Application Server system messages by using the Installed Application Output field of the 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 a UNIX system.

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
Specify the number of hours, from 1 to 24, when a periodic rollover occurs.

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 the JVM, may write data to these process streams. In addition, the JVM provided System.out and System.err streams can be configured to write their data to these streams also.

By default, the stdout and stderr streams are redirected to log files at application server startup, which contain text written to the stdout and stderr streams by native modules (.dlls, .exes, UNIX libraries, and other modules). By default, these files are stored as *installation_root*/logs/*applicationServerName*/native_stderr.log and native_stdout.log.

Viewing the service log

The service log is a special log written in a binary format. You cannot view the log directly using a text editor. You should never directly edit the service log, as doing so will corrupt the log. To move the service log from one machine to another, you must use a mechanism like FTP, which supports binary file transfer.

You can view the service log in two ways:
- It is recommended that you use the Log Analyzer tool to view the service log. This tool provides interactive viewing and analysis capability that is helpful in identifying problems.
- If you are unable to use the Log Analyzer tool, you can use the Showlog tool to convert the contents of the service log to a text format that you can then write to a file or dump to the command shell window. The steps for using the Showlog tool are described in "Showlog Script."

Showlog Script

The service log is a special log written in a binary format. You cannot view the log directly using a text editor. You should never directly edit the service log, as doing so will corrupt the log. To move the service log from one machine to another, you must use a mechanism like FTP, which supports binary file transfer.

You can view the service log by using the Showlog script 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.

To run the showlog script:
1. Open a shell window on the machine where the service log resides.
2. Change the directory to *install_directory*/bin where *install_directory* is the fully qualified path where the WebSphere Application Server product is installed.
3. Run the showlog script:

   On Windows systems, the script is named showlog.bat.

Displaying user instructions

Run the showlog script with no parameters to display usage instructions. On Windows systems, the script is named

```
showlog.bat
```

Displaying the service log contents to the shell window

Use the invocation

```
showlog service_log_filename
```

If the service log is not in the default location, you must fully qualify the service_log_filename

Format and write the service log contents to a file

Use the invocation

```
showlog service_log_filename output_filename
```

If the service log is not in the default location, you must fully qualify the service_log_filename

Using the showlog command to view Common Base Events

The showlog command converts the system activity.log file from binary format into plain text. Use the following showlog commands to produce output in Common Base Event XML format.

1. On a Windows platform: showlog -format=CBE-XML-1.0.1 *filename*
2. On a UNIX platform: showlog.sh -format=CBE-XML-1.0.1 *filename*

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 *install_directory*/logs/activity.log. If the name is changed, the run time requires write access to the new file, and the file must use the .log extension.
5. Set the maximum file size. Specifies the number of megabytes to which the file can grow. When the file reaches this size, it wraps, replacing the oldest data with the newest data.
6. Set the message filter level to the desired state.
7. Save the configuration.
8. Restart the server to apply the configuration changes.

IBM service log settings

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 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.

Message Filtering:

This setting has no effect. Log level details can now be used to control both message and trace filtering.

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.

Detecting hung threads in J2EE applications

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.

When WebSphere Application Server 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 System.Out log file 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. See “Detecting hung threads in J2EE applications.”
- 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.

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 System.out 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 System.out 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"

Adjusting the hang detection policy of a running server

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 ${threshold}
```

Configuring the hang detection policy

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. (Link to
       Detecting hung threads in J2EE applications for information on false alarms)
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.

Working with trace

Use trace to obtain detailed information about the execution of WebSphere Application Server components, including application servers, clients, and other processes in the environment. Trace files show the time and sequence of methods called by WebSphere Application Server base classes, and you can use these files to pinpoint the failure.

Collecting a trace is often requested by IBM technical support personnel. If you are not familiar with the internal structure of WebSphere Application Server, the trace output might not be meaningful to you.

1. Configure an output destination to which trace data is sent.
2. Enable trace for the appropriate WebSphere Application Server or application components.
3. Run the application or operation to generate the trace data.
4. Analyze the trace data or forward it to the appropriate organization for analysis.

Enabling tracing and logging

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 version 6, a new 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 Servers. 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 version 6, 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 version 5.1.1 and earlier, trace strings were used for configuring tracing only. In WebSphere Application Server version 6.0, the ″trace string″ becomes a ″logging string″; it is used to configure both tracing and message logging.

In WebSphere Application Server version 5.1.1 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 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 version 5.1.1 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 version 6.0, 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 ″*″ is used by itself in place of the component name, the level the string specifies, will be applied to all components.

Note:

- In WebSphere Application Server version 5.1.1 and earlier, you could set the level to ″all=disabled″ to disable tracing. This syntax in version 6 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 version 6, ″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. 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 Trace** 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.
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 68 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 > Logging and Tracing** in the console navigation tree, then click **server > Diagnostic Trace**.
3. Select the **Runtime** tab.
4. Select the **Save Trace** check box if you want to write your changes back to the server configuration.
5. Change the existing trace state by changing the trace specification to the desired state.
6. Configure the trace output if a change from the existing one is desired.
7. Click **Apply**.

Enabling trace on client and standalone applications

When standalone client applications (such as Java applications which access EJBs 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 `DtraceSettingsFile=`*filename* system property 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 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 *install_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.
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.

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.

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.
   c. To change the format in which trace output is generated, select the appropriate value from the drop-down list.

Interpreting trace output

On an application server, trace output can be directed either to a file or to an in-memory circular buffer. If trace output is directed to the in-memory circular buffer, it must be dumped to a file before it can be viewed.

On an application client or 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

Specifying the log analyzer format allows you to open trace output using the Log Analyzer. This is useful if you are trying to correlate traces from two different server processes, because it allows you to use the merge capability of the Log Analyzer).

Diagnostic trace service settings

To view this page, click **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.

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
Preserves trace information in a format that is compatible with the Log Analyzer tool. Select this option if you want to use the trace output as input to the Log Analyzer tool.

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.

Log and trace settings

Use this page to view and configure logging and trace settings for the server.

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** *server_name* > *service_type*.

Name

Specifies the logical name of the server. For WebSphere Application Server for z/OS, this is sometimes called the long name.

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.

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. Before opening a PMR, see the IBM Support page.

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. The collector tool can be executed 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:

To run the collector tool:

1. Log on to the system as **root** (or **Administrator** in a Windows platform).
2. Verify that Java 1.2.2 or higher is available in the path. The collector program requires Java code to run. It also collects data about the IBM Developer Kit, Java Technology Edition in which it runs. If there are multiple Developer Kits on the system, verify that the one that the WebSphere Application Server product uses is the one in the path for the collector program. If the Developer Kit being used by the WebSphere Application Server is not available, putting another Developer Kit in the path for the collector program lets you collect all data except information about the Developer Kit.
3. Verify that all necessary information is in the path being used by the collector program and that you are not running the program from within the WebSphere Application Server product installation root directory.
   a. If this system is a Linux or UNIX-based platform, verify that the path contains:
      - `/bin`
      - `/sbin`
      - `/usr/bin`
      - `/usr/sbin`

   b. If this system is a Windows platform, include `regedit` in the path.
4. Create a work directory where you can start the collector program.
5. Make the work directory the current directory. 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 WebSphere Application Server installation directory.
6. Run the collector program by entering the command: **`collector`** from the command line.

   Using the **collector** command with no additional parameters gathers one copy of the node data and data from each server in the node, and stores them in a single JAR output file. To gather data from a specific server in the node, use the command **`collector.bat/sh -servername`** ***`servername`***, where *`servername`* is the name of the problem server.

   Note:

   Set the path correctly to use the non-qualified version of the command. For Linux and UNIX-based platforms, *`install_root`*`/bin` must be in the path to locate the **collector.sh** command. For Windows platforms, *`install_root`*`\bin` must be in the path to locate the **collector.bat** command.

   The WebSphere Application Server installation root directory is determined at installation. It is identified in the `setupCmdLine.sh` file (or the `setupCmdLine.bat` file on a Windows platform).

   You can enter a fully qualified path to the collector command. For example, enter this command in a default installation on a Windows platform:

   ```
   c:\WebSphere\AppServer\bin\collector.bat
   ```

The collector program creates a log file, `Collector.log`, and an output JAR file in the current directory.

The name of the JAR file is based on the hostname and package of the Application Server product, in the format: *hostname-cellname-nodename-profile*.

The `Collector.log` log file is one of the files collected in the *hostname-cellname-nodename-profile* file.

Send the *hostname-cellname-nodename-profile* 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
- Operating system information in a directory named `OS`
- Java information in a directory named `Java`
- WebSphere Application Server information in a directory named `WAS`
- Collector shell script (or batch file) execution information in a directory named `debug`
- MQ information in a directory named `MQ`, if you installed WebSphere MQ or the embedded messaging feature
- 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.
- On Linux and UNIX-based systems, the file `OS/commands` has the location of all commands used. If you are missing command output, check this file to see if the command was found.
- On Linux and UNIX-based systems, the collector runs some shell scripts. The shell script output is saved in files in the `OS` directory, while the corresponding debug information is saved in the `debug` directory. If the output of a shell script is missing, check the corresponding file in the `debug` directory.
- When you issue the **collector** command when there are multiple installation instances, the tool that runs depends on what is in the `PATH` statement. For example, if you install both the base WebSphere Application Server and the Deployment Manager product on the same machine, the `bin` directory that first appears in the `PATH` variable is the one that furnishes the collector tool. To work around this problem, use a fully qualified file path when calling the collector tool as shown in this example for a Windows platform:

  ```
  c:\WebSphere\AppServer\bin\collector.bat
  ```

- On Windows systems, the OS directory contains a file named installed.out. This file contains a list of programs found in the Add/Remove Programs list. This same information is contained in the file Desktop\My Computer\Control Panel\Add/Remove Programs\Install/Uninstall.

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.

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.

Linux and UNIX-based platforms:

install_root/bin/collector.sh -Summary

Windows platforms:

install_root\bin\collector.bat -Summary

First Failure Data Capture tool

The First Failure Data Capture tool preserves the information generated from a processing failure and returns control to the affected engines. The captured data is saved in a log file for use in analyzing the problem.

The First Failure Data Capture tool is intended primarily for use by IBM Service. It runs as part of the IBM WebSphere Application Server, and you cannot start or stop it. It is recommended that you not attempt to configure the First Failure Data Capture tool. If you experience conditions requiring you to contact IBM Service, your IBM Service representative will assist you in reading and analyzing the First Failure Data Capture log.

The First Failure Data Capture tool does not affect the performance of the IBM WebSphere Application Server.

Log Analyzer

The Log Analyzer takes one or more service or activity logs, merges all of the data, and displays the entries. Based on its symptom database, the tool analyzes and interprets the event or error conditions in the log entries to help you diagnose problems. Log Analyzer has a special feature enabling it to download the latest symptom database from the IBM Web site.

To download the latest updates to the symptom database, use the **File -> Update Database -> WebSphere Application Server Symptom Database** option for WebSphere Application Server, or

WebSphere Application Server Network Deployment Symptom Database option for WebSphere Application Server Network Deployment in the Log Analyzer interface.

About the service or activity log

The application server creates the service or activity log file from the activity of the various WebSphere Application Server components. Log Analyzer is used to view the service or activity log file. Log Analyzer can merge service or activity log files into one log file. The service or activity log file, `activity.log`, is a binary file in the `logs` directory of the install_root.

You cannot view the service or activity log with a text editor. The Log Analyzer tool lets you view the file.

Viewing a service or activity log file in the absence of a graphical interface

The Log Analyzer tool cannot view remote files. If the operating system on which you are running WebSphere Application Server does not support the use of a graphical interface, transfer the file in binary mode to the system on which you are running the Java administrative console. Use the Log Analyzer tool there.

In cases where transferring the file is impractical or inconvenient, use the alternate viewing tool, **showlog**, to view the service or activity log file:
1. Change directory to `bin` directory of the install_root.
2. Run the **showlog** tool with no parameters to display usage instructions:
   - On Windows systems, run **showlog.bat**.
   - On UNIX systems, run **showlog.sh**.

To direct the service or activity log (`activity.log`) contents to stdout, use the **showlog activity.log** command.

To dump the service or activity log to a text file for viewing with a text editor, use the **showlog activity.log** *textFileName* command.

Accessing Log Analyzer help files

You can access Log Analyzer help files on Windows platforms, using the operating system default Internet browser only. You cannot access the help files using an Internet browser other than the default. However, Windows systems do let you select either Netscape or Internet Explorer as the default browser. There is an option to let you select either Netscape or Internet Explorer as the browser to display HTML help files.

Access help files using any Internet browser on UNIX platforms. You can use such browsers as Netscape Navigator, by explicitly setting the location of its executable in the tool Preferences dialog. The option that appears to allow you to select either Netscape or Internet Explorer as the browser to display HTML help files is not used on UNIX systems.

To specify the browser on UNIX platforms:
1. Click **File > Preferences** in the Log Analyzer tool.
2. Click **Help** from the **General** folder in the Log Analyzer Preferences dialog.
3. Set the path to the Internet browser executable in the Browser Location field.

Installing Log Analyzer silently

Installing Log Analyzer ″silently″ prevents installation messages from being displayed, but the file `responsefile.txt` for silent installation needs more information to install Log Analyzer. To silently install Log Analyzer, add the following option to this file:

```
-P logAnalyzerBean.active="true"
```

The Performance and Analysis Tools property in the file `responsefile.txt` needs to be set to `true` to install the Log Analyzer tool. The property in the `responsefile.txt` is: -P performanceAndAnalysisToolsBean.active=″true″.

Using the Log Analyzer

To view the service or `activity.log` using the Log Analyzer:

1. Change directory to: *install_dir*/bin.
2. Run the waslogbr script file. This file is named:
   - `waslogbr.bat` on Windows systems.
   - `waslogbr` on UNIX systems.

   This script must be run from the *install_dir*/bin directory. This starts the Log Analyzer interface.
3. Select **File -> Open**.
4. Navigate to the directory containing the service or activity log file.
5. Select the service or activity log file and click **Open**.
6. To analyze the records, right-click on a record in the tree on the left, select **UnitOfWorkView** from the right-click menu, and select **Analyze**. Now any records with a green check mark next to them match a record in the symptom database. When you select a check-marked record, you will see an explanation of the problem in the lower-right-hand pane.

   Note: When starting the Log Analyzer for the first time, or after the Log Analyzer preferences files of the users have been deleted, the following message is displayed in the Log Analyzer's shell window:

   ```
   Cannot open input stream for waslogbrsys
   ```

   You can disregard this message, as it is informational and not indicative of abnormal operation.

   WebSphere Application Server includes the following Log Analyzer files for use with the WebSphere Commerce Suite:
   - install_root\bin:
     - wcslogbr.bat
     - wcslogbrsys.cfg
     - wcslogbrsys.ini
   - install_root\properties\logbr:
     - wcsanalyzers.xml
     - wcslogtypes.xml
     - wcsrecdef.xml

   You can ignore these files.

Log Analyzer main window: To view this page, launch the Log Analyzer, install_root/bin/waslogbr on UNIX systems or install_root\bin\waslogbr.bat on Windows NT or Windows 2000 systems. Click **Help > Tasks**.

The Log Analyzer takes one or more service or activity logs, merges all the data, and, by default, displays the entries in unit of work (UOW) groupings. It analyzes event and error conditions in the log entries to provide message explanations. The Log Analyzer main window interface has the following elements:
- Three window panes
- Status line
- Menu bar
- Pop-up actions

Window panes

The Log Analyzer window has three panes:
Logs pane (left)

By default, Log Analyzer Logs pane displays log entries by UOW. It lists all the UOW instances and its associated entries from the logs that you have opened. You may find the UOW grouping useful when you are trying to find related entries in the service or activity log or when you are diagnosing problems across multiple machines. The file name of the first log you open appears in the pane title bar. There is a root folder and under it, each UOW has a folder icon which you can expand to show all the entries for that UOW. All log entries without any UOW identification are grouped into a single folder in this tree view. The UOW folders are sorted to show the UOW with the latest timestamp at the top of the list. The entries within each UOW are listed in the reverse sequence, that is the first (earliest) entry for that UOW is displayed at the top of the list. If you have merged several logs in the Log Analyzer, all the log entries are merged in timestamp sequence within each UOW folder, as if they all came from the same log.

Every log entry is assigned an entry number, Rec_*nnnn*, when a log is opened in the Log Analyzer. If more than one file is opened in the Log Analyzer (merged files), the Rec_*nnnn* identification will not be unique because the number is relative to the entry sequence in the original log file and not to the merged data that the Log Analyzer is displaying. This Rec_*nnnn* appears in the first line (**RecordId**) in the Records pane.

By default, each entry in this pane is color-coded to help you quickly identify the ones that have high severity errors. The values listed here are the default values, you can configure your own colors.

- Non-selected log entry with background color of:
  - Pink indicates that it has a severity 1 error.
  - Yellow indicates that it has a severity 2 error.
  - White indicates that it has a severity 3 error.
- Selected log entry with background color of:
  - Red indicates that it has a severity 1 error.
  - Green indicates that it has a severity 2 error.
  - Blue indicates that it has a severity 3 error.

These colors are configurable and can be changed in the Log Analyzer Preferences Log page. See the help for the Severity page in the Log Analyzer Preferences notebook for different error severity levels and for more information on how to do this.

The Log Analyzer can display the log entries in different groupings. Use the Log Analyzer Preferences notebook: Logs page to set the grouping filters.

After the Analyze action has been invoked, each analyzed log entry has the following icons:

- A check icon indicates that the entry has some analysis information in one or more pages in the Analysis pane.
- A cascading plus sign (+) icon indicates that the entry has some analysis information and that it has a reraised or remapped exception. You may want to look at the log entry prior to this one when diagnosing problems.
- A question mark icon indicates that the entry has either a severity 1 or 2 error but no additional analysis information is available for it.
- An ″x″ icon indicates that the entry has a severity 3 error and it has no analysis information.

Record pane (upper right)

When you select an entry in the Logs pane, you see the entry in the Record pane. The entry identification appears in the pane title bar. Right-click in the Record pane to see actions that you can perform on the selected entry. A drop down arrow next to Record lets you look at the last ten records you have viewed. The cache for the historical data (10, by default) is set in the Preferences General page.

Note:

- The page does not display associated analysis data for these cached records. To see analysis information for cached data, reselect the entry from the Logs pane.

You can enable or disable line wrap mode for the Record Pane using the Log Analyzer Preferences notebook: Record. To print contents of this pane, select **Record > Print** when the Record pane is in focus.

Analysis pane (lower right)

When the analyze action has been invoked and additional information is available, the information will appear in the Symptom page. If the page tab is grayed out, there is no information in that page. The pages of the Analysis pane are:

Symptom

The Log Analyzer provides a database of information on common events and errors to help you recover from some common errors. As a part of the analyze action, if such information is found in the database for the selected log entry, the information is displayed in this page.

Status line

There is a status line at the bottom of the window showing the status of actions.

Menu bar

The menu bar in the Log Analyzer main window, has the following selections:

File

Open...

Opens a new log file. You can select either a service or activity log or a previously saved XML file. If you want the Log Analyzer to format a raw log file (by running the showlog command) prior to opening it, name the log file with suffix.log. If the Log Analyzer finds that the .log file contains formatted data, it skips the showlog formatting step.

If you want to merge data from another log, select **Merge with**.

Merge with...

When another log file is already opened in the Log Analyzer, use the **Merge with** action to open subsequent logs. The Log Analyzer merges the data from all the logs that it opens and displays all the entries within timestamp sequence in the UOW folders. The data appears as if they came from one log.

If you want the Log Analyzer to format a raw log file (by running the showlog command) prior to opening it, name the log file with suffix.log. If the Log Analyzer finds that the .log file contains formatted data, it skips the showlog formatting step.

Redisplay logs

To redisplay the logs using the recently set filters.

Save as...

Saves the log as an XML file (or text file). If **analyze action** has been performed, all the Symptom analysis information is also saved. If logs are merged in the Log Analyzer, the saved file contains entries of all the merged logs in the sequence that is shown in the Logs pane.

Note: If the merged logs have different timestamp formats, you should not save the merged information because the Log Analyzer only recognizes a single timestamp format for each file that it opens.

Save

Is only enabled if the first file that you opened is an XML file. It resaves the XML file with all the data that is currently displayed in the Log Analyzer. If **analyze action** has been performed, all the Symptom analysis information is also saved. If logs are merged in the Log Analyzer, the saved file contains entries of all the merged logs in the sequence that is shown in the Logs pane.

Note: If the merged logs have different timestamp formats, you should not save the merged information because the Log Analyzer only recognizes a single timestamp format for each file that it opens.

Print Log...
Prints all the entries that the Log Analyzer is displaying. If logs are merged in the Log Analyzer, the output contains entries of all the merged logs in the sequence that is shown in the Logs pane. If analyze action has been performed, all Symptom analysis information is also printed. To print parts of the log, use **Record > Print**.

Close Closes the opened log.

Update Database
Updates the symptom database which is used for Symptom analysis. It downloads the latest version of the symptom database from the URL specified in the ivblogbr.properties file.

Preferences...
Lets you configure and change the appearance of the Log Analyzer window and its contents.

Exit Exits the Log Analyzer and closes its window.

Edit

Copy Copies the selected text in the Record or Analysis pane to the clipboard. If you have not selected any text, **Copy** does not appear in the menu.

Find Allows you to find text strings in the focused pane.

View

Logs Toggles the visibility of the Logs pane.

Record
Toggles the visibility of the Record pane.

Symptom
Toggles the visibility of the Symptom page in the Analysis pane.

Record

All the actions under this menu applies to the focused pane.

To select several entries, hold down the **Ctrl** key when making the selection. When a folder is selected, the action applies to all the entries in that folder.

Analyze

Retrieves and displays additional documentation on known events and event messages in the Analysis pane (Symptom page). Select the folders and entries in the Logs pane, right-click to select the **Analyze** action, or from the menu bar, select **Record > Analyze**.

Note: If you invoke Analyze for the root folder, then all the entries in the log that you are viewing will be analyzed.

If some analysis information is available for an entry, it will either have a check icon or a cascading plus sign (+) icon next to it in the Logs pane. If the analyze action has already been performed, the selection will be grayed out.

Save to file
Saves the selected entries in the Logs pane. If folders are selected, all the entries in the folder are saved. Any retrieved analysis information is also saved. If the focused pane is either the Records or Analysis pane, then only information in that pane is saved.

Print

- If the focused pane is Logs, the action prints the selected folders and entries. Any retrieved analysis information for those entries is also printed.
- If the focused pane is Record, the action prints the entry that is currently in the Record pane. Any retrieved analysis information is not printed.
- If the focused pane is Analysis, the action prints Symptom page contents.

Windows
If you detach the Symptom page in the Analysis pane into separate windows, all windows appear under this menu. You can select windows to bring them to the foreground.

Help Provides a list of online documentation for additional information.

Pop-up actions

In the focused pane, right-click to bring up a list of actions in a pop-up menu. Actions that you cannot perform are grayed out. When a folder is selected in the Logs pane, the action applies to all entries in that folder. To select several folders or entries in the Logs pane, hold down the **Ctrl** key when making the selection.

Log Analyzer find window: To view this page, launch the Log Analyzer, install_root/bin/waslogbr on UNIX systems or install_root\bin\waslogbr.bat on Windows NT or Windows 2000 systems. Click **Edit > Find**.

The Log Analyzer Find window lets you look for text strings in the focused pane. For example, if you remember the Unit of Work identification, you can enter that text string in the Find window to quickly locate the Unit of Work folder in the Logs pane.

Log Analyzer Preferences notebook - General: To view this page, launch the Log Analyzer, install_root/bin/waslogbr on UNIX systems or install_root\bin\waslogbr.bat on Windows NT or Windows 2000 systems. Click **File > Preferences > General**.

The General page of the Log Analyzer Preferences notebook lets you specify the behavior of panes in the Log Analyzer window:

Show title bars
Shows the title bars of window and its panes.

Highlight selected pane
Highlights the pane that is in focus.

Pane history cache size
Specifies a number of records to save in the cache. The Log Analyzer keeps a history of the (specified number of) records that you have viewed. You can use the drop down list next to Record in the pane title bar to see these cached entries.

Note: The associated analysis data for these records are not saved. To see analysis information, reselect the entry from the Logs pane.

Show logo at startup
Shows the logo when you start-up the Log Analyzer.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Log Analyzer Preferences notebook - Appearance: To view this page, launch the Log Analyzer, install_root/bin/waslogbr on UNIX systems or install_root\bin\waslogbr.bat on Windows NT or Windows 2000 systems. Click **File > Preferences > General > Appearance**.

The Appearance page of the Log Analyzer Preferences notebook lets you define the overall appearance of the Log Analyzer. You can select the family of products and its texture schemes that you want the Log Analyzer window to emulate.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Log Analyzer Preferences notebook - Toolbars: To view this page, launch the Log Analyzer, install_root/bin/waslogbr on UNIX systems or install_root\bin\waslogbr.bat on Windows NT or Windows 2000 systems. Click **File > Preferences > General > Toolbars**.

The Toolbars page of the Log Analyzer Preferences notebook lets you customize the appearance and contents of the toolbar in the Log Analyzer window. You can select whether there is text and/or icon in the toolbar, as well as, the functions that you want in the toolbar.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Log Analyzer Preferences notebook - Help: To view this page, launch the Log Analyzer, install_root/bin/waslogbr on UNIX systems or install_root\bin\waslogbr.bat on Windows NT or Windows 2000 systems. Click **File > Preferences > General > Help**.

The Help page of the Log Analyzer Preferences notebook lets you select the browser that is to display online help files.

For Windows, the default Web browser is used. You need not update any settings unless there are problems when bringing up the default browser.

For AIX, HP-UX, and Solaris operating environment, you must update the following settings, especially the full path of the browser in the **Browser location** entry.

Help browser
Select the Web browser you want to use.

Browser location
Select the location of the browser executable file. This should be correct by default, but if you cannot access help then you may need to explicitly enter the browser location.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Log Analyzer Preferences notebook - Proxy: To view this page, launch the Log Analyzer, install_root/bin/waslogbr on UNIX systems or install_root\bin\waslogbr.bat on Windows NT or Windows 2000 systems. Click **File > Preferences > Proxy**.

The symptom database included in the Log Analyzer package contains entries for common events and errors. New versions of the symptom database provide additional entries. Download new versions of the database from the IBM FTP site. The URL for the FTP site is located in file: *install_dir*/bin/ivblogbr.properties.

The default setting for the FTP site is:
ftp://ftp.software.ibm.com/software/websphere/info/tools/loganalyzer/symptoms/std/symptomdb.xml

You can update your symptom database in one of two ways:
1. Download a new version from the FTP site, and replace your existing database with the new version. Your database is: *install_dir*/symptoms/std/symptomdb.xml.
2. Use the Log Analyzer graphical user interface (GUI) to update your database by selecting: **File -> Update database -> WebSphere Application Server Symptom Database** (for WebSphere Application Server) or **WebSphere Application Server Network Deployment Symptom Database** for WebSphere Application Server Network Deployment.

Setting the proxy definition

If your organization uses a FTP or SOCKS proxy server, contact your system administrator for the host name and port number of the proxy server.

If you use the Log Analyzer GUI to update the database, you can add a proxy definition to the Proxy Preferences page as described below:
1. Select **File -> Preferences -> Proxy**.
2. Select the appropriate proxy type.
3. Enter the host name and port number of the proxy server on the **Proxy** panel.

If you do not use the Log Analyzer GUI, add the proxy definition to the command that launches Log Analyzer.
- Do the following to add the proxy definition for the FTP proxy server:

- For Windows systems:
  1. Modify file: *install_dir*\bin\waslogbr.bat.
  2. Add the following text to the file:

```
%JAVA_HOME%\bin\java -DIVB_HOME=%USERPROFILE%/logbr ^
....
-Dftp.proxyHost=proxy_host -Dftp.proxyPort=port_number ^
```

- For UNIX:
  1. Modify file: *install_dir*/bin/waslogbr.
  2. Add the following text to the file:

```
$JAVA_HOME/bin/java -ms10m -mx255m -DIVB_HOME=$HOME/logbr \
....
-Dftp.proxyHost=proxy_host -Dftp.proxyPort=port_number \
```

- Do the following to add the proxy definition for the SOCKS proxy server:
  - For Windows systems:
    1. Modify file: *install_dir*\bin\waslogbr.bat.
    2. Add the following text to the file:

```
%JAVA_HOME%\bin\java -DIVB_HOME=%USERPROFILE%/logbr ^
....
-DsocksProxyHost=proxy_host -DsocksProxyPort=port_number ^
```

  - For UNIX:
    1. Modify file: *install_dir*/bin/waslogbr.
    2. Add the following text to the file:

```
$JAVA_HOME/bin/java -ms10m -mx255m -DIVB_HOME=$HOME/logbr \
....
-DsocksProxyHost=proxy_host -DsocksProxyPort=port_number \
```

Log Analyzer Preferences notebook -- Logs: To view this page, launch the Log Analyzer, user_install_root/bin/waslogbr on UNIX systems or user_install_root\bin\waslogbr.bat on Windows NT or Windows 2000 systems. Click **File > Preferences > Logs**.

The Logs page of the Log Analyzer Preferences notebook lets you group the entries in the logs by different entry fields for viewing. For example, you can select to group the log entries by TimeStamp or clientHostName when they are displayed in the Logs pane.

Primary sort field

Use this filter to set the first level of grouping when log entries are displayed in the Logs pane. By default, the log entries are grouped by UnitOfWork.

Secondary sort field

Use this filter to set the second level of grouping (that is, within the grouping of the primary sort field) when log entries are displayed in the Logs pane.

All the entries within the grouped folders are always sorted in timestamp sequence with the earliest entry at the top of the list.

Redisplay log file immediately

Select this box to immediately regroup the logs entries (after you have clicked **OK**) based on the new filter settings. The entries in the Logs Pane are redisplayed according to the new grouping. If you want to delay the grouping, then do not select this box and, at a later time, you can use the **File > Redisplay logs...** menu selection to regroup and display the log entries based on the changed filter settings.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Log Analyzer Preferences notebook -- Severity: To view this page, launch the Log Analyzer, install_root/bin/waslogbr on UNIX systems or install_root\bin\waslogbr.bat on Windows NT or Windows 2000 systems. Click **File > Preferences > Logs > Severity**.

The Severity page of the Log Analyzer Preferences notebook lets you change background colors of log entries that appear in the Logs pane. Use colors to quickly indicate entries with high severity errors and the currently selected entry.

Use colors to indicate severities

Select this checkbox to color-code the background of log entries and folders. When selected, the radio button selections in this page are enabled.

Background color

For each folder and entry in the Logs pane, there is some text describing the entry. To choose a background color for selected log entry that has a severity 1 error, do the following:

1. Select **Selected node**.
2. Select **Severity 1**.
3. Select the color by clicking on the color Swatches. To use the default setting, click **Restore Default**. To see the results of your change, look at the Preview box.
4. Click **Apply** to save that setting.

Repeat similar steps to change the background color for selected log entries that have severity 2 and 3 errors.

To choose a background color for an unselected log entry that has a severity 1 error, do the following:

1. Select **Unselected node**.
2. Select **Severity 1**.
3. Select the color by clicking on the color Swatches. To use the default setting, click **Restore Default**. To see the results of your change, look at the Preview box.
4. Click **Apply** to save that setting.

Repeat similar steps to change the background color for unselected log entries that have severity 2 and 3 errors.

Sample

You can see the result of you color change prior to applying the change. Look at the nodes shown in the Sample box. For color changes of selected nodes, click on the node in the sample box to see the color change.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Log Analyzer Preferences notebook -- Analyzer output: To view this page, launch the Log Analyzer, install_root/bin/waslogbr on UNIX systems or install_root\bin\waslogbr.bat on Windows NT or Windows 2000 systems. Click **File > Preferences > Analyzer output**.

The Log Analyzer Preferences notebook lets you enable line wrap for information that appears in the Analysis pane.

Set line wrap

Select the appropriate checkbox to enable line wrap for the Symptom page that appears in the Analysis pane.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Log Analyzer Preferences notebook -- Record: To view this page, launch the Log Analyzer, install_root/bin/waslogbr on UNIX systems or install_root\bin\waslogbr.bat on Windows NT or Windows 2000 systems. Click **File > Preferences > Record**.

The Record page of the Log Analyzer Preferences notebook lets you set the line wrap mode for the data that is displayed in the Record pane. It also lets you set the time and date format of the timestamp displayed in the Record pane.

Enable line-wrap mode for record pane

Select this checkbox to enable line wrapping for the information that appears in the Record pane.

Date format

When viewing logs, this date format only changes the timestamp format that is displayed in the Record pane. The timestamp format in the log file and the timestamp shown in the Logs pane are not affected by this setting.

Time format

When viewing logs, this time format only changes the timestamp format that is displayed in the Record pane. The timestamp format in the log file and the timestamp shown in the Logs pane are not affected by this setting.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Installing the Log Analyzer silently:

To silently install the Log Analyzer, you must set the relevant properties in the `responsefile.txt` file.

1. Add the following option to the `responsefile.txt` file:
   - `-P logAnalyzerBean.active="true"`
2. Set the Performance and Analysis Tools property in the `responsefile.txt` file to true for the Log Analyzer to install. The property in the responsefile.txt file is:
   - `-P performanceAndAnalysisToolsBean.active="true"`

Accessing the Log Analyzer help files:

For Windows platforms, you can only access the Log Analyzer help files using the operating system default Internet browser. You cannot access the help files using any Internet browser, even though there are options allowing you to select either Netscape or Internet Explorer and set the location of the browser to display HTML help files.

For UNIX platforms, you can access the help files using any Internet browser, such as Netscape Navigator, by explicitly setting the location of the browser executable in the tool Preferences dialog. The option that seemingly allows you to select either Netscape or Internet Explorer as the browser to display HTML help files is not used.

The following steps describe how to specify the browser on UNIX platforms:

1. In the Log Analyzer tool, select **File -> Preferences**
2. In the Log Analyzer **Preferences** dialog, click **Help** from the**General** folder.
3. Set the path to the Internet browser executable in the **Browser Location** field.

IBM Support Assistant

The IBM Support Assistant is a tool that helps you use various IBM Support resources from within the WebSphere Application Server administrative console. The IBM Support Assistant offers three 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 IBM 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 service request that includes key system data to IBM.

To download the IBM Support Assistant:

1. Log into the administrative console and select **Support** from the top menu. This will open the product support page, from where you can download the IBM Support Assistant and also access various IBM Web resources such as IBM product sites, IBM support sites and links to IBM news groups.
2. Select ″Click here to download the IBM Support Assistant″ to download the tool and follow the instructions in the README file to install the tool.

Once the IBM Support Assistant is installed in the administrative console, you can launch it by selecting 'Support' from the top menu in the administrative console. The search component of the IBM Support Assistant will be displayed in a new browser window.

To learn more about how to use the IBM Support Assistant, click on the User Guide tab in the IBM Support Assistant window.

Note: The Support link in the administrative console is a context sensitive link. When the IBM Support Assistant is not installed, the Support link displays a link to the IBM Support Assistant download page and various IBM Web resources for the WebSphere Application Server product. Once the IBM Support Assistant is installed, the Support link launches the IBM Support Assistant main page.

Obtaining help from IBM

If you are not able to resolve a WebSphere Application server problem by following the steps described in the Troubleshooting guide, by looking up error messages in the message reference, or looking for related documentation on the online help, contact IBM Technical Support.

Purchase of WebSphere Application Server entitles you to one year of telephone support under the Passport Advantage program. For details on the Passport Advantage program, visit 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.

The Collector Tool

WebSphere Application Server comes with a built-in utility that collects logs and configuration information into one file, the Collector Tool. IBM Technical Support may ask you to run this tool and submit the output.

Tracing

WebSphere Application Server support engineers might ask you to enable tracing on a particular component of the product to diagnose a difficult problem. For details on how to do this, see Enabling trace.

Consulting

For complex issues such as high availability and integration with legacy systems, education, and help in getting started quickly with the WebSphere product family, consider using IBM consulting services. To learn about these services, browse the Web site http://www-1.ibm.com/services/fullservice.html.

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.
- *Hints and Tips*, *Technotes*, and *Solutions* links take you to specific problems and resolutions documented by WebSphere Application Server technical support personnel.
- A link *All fixes, fix packs, refresh packs, and tools* provides free WebSphere Application Server maintenance upgrades and problem determination tools.
  - *fixes* are software patches which address specific WebSphere Application Server defects. Selecting a specific defect from the list in the *All fixes, fix packs, refresh packs, and tools* 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 *list defects* 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.
  - Tools are free programs that help you analyze the configuration, behavior and performance of your WebSphere Application Server installation.

Accessing WebSphere Application Server support page resources

Some resources on the WebSphere Application Server support page are marked with a key icon. To access these resources, you must supply a user ID and password, or to register if do not already have an ID. When registering, you are asked for your contract number, which is supplied as part of a WebSphere Application Server purchase.

WebSphere Developer Domain

The Developer Domains are IBM-supported sites for enabling developers to learn about IBM software products and how to use them. They contain resources such as articles, tutorials, and links to newsgroups and user groups. You can reach the WebSphere Developer Domain at
http://www7b.software.ibm.com/wsdd/.

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.

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.

BSF debug port

Specifies the port that the BSF Debug Manager listens on.

BSF logging level

Specifies the level of logging provided by the BSF Debug Manager. The valid range is 0-3, with 3 being the highest level of logging.

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.

Click a configuration problem in the Configuration Problems table to see more information about the problem.

Configuration document validation

Use these fields to specify the level of validation to perform on configuration documents.

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.

Fields that explain each configuration problem:

The following informational fields appear when you click a configuration problem message.

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, with 1 being a severe error. Severity decreases as the severity descriptor increases.

Local URI
Specifies the local URI of the configuration file where the error occurred.

Full URI
Specifies the full URI of the configuration file where the error occurred.

Validator classname
The classname of the validator reporting the problem.

Chapter 6. Learn about WebSphere applications

Use this section as a starting point to investigate the technologies used in and by applications that you deploy on the application server.

See Learn about WebSphere applications: Overview and new features for an introduction to each technology.

Web applications	How do I?...	Overview		Samples
EJB applications	How do I?...	Overview	Tutorials	Samples
Client applications	How do I?...	Overview		Samples
Web services	How do I?...	Overview	Tutorials	Samples
Data access resources	How do I?...	Overview	Tutorials	Samples
Messaging resources	How do I?...	Overview	Tutorials	Samples
Mail, URLs, and other J2EE resources	How do I?...	Overview		
Security	How do I?...	Overview	Tutorials	Samples
Naming and directory	How do I?...	Overview		
Object Request Broker	How do I?...	Overview		
Transactions	How do I?...	Overview		Samples
ActivitySessions	How do I?...	Overview		Samples
Application profiling	How do I?...	Overview		Samples
Asynchronous beans	How do I?...	Overview		Samples
Dynamic caching	How do I?...	Overview		
Dynamic query	How do I?...	Overview		Samples
Internationalization	How do I?...	Overview		Samples
Object pools	How do I?...	Overview		
Scheduler	How do I?...	Overview		Samples
Startup beans	How do I?...	Overview		
Work areas	How do I?...	Overview		

Web applications

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 or 6.

Symptom Your Web module does not run when you migrate it to Version 5 or 6

Problem In Version 4.x, the classpath setting that affected visibility was *Module Visibility Mode*. In Versions 5 and 6, 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.

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 article Assembling Web applications for more information.

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.

EJB applications

Access intent exceptions

The following exceptions are thrown in response to the application of access intent policies:

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

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 databases, the weakest lock is a shared one, and the query runs without a USE AND KEEP UPDATE LOCKS clause. On Oracle databases, however, the weakest lock is an update lock; this means that the SQL query must contain a USE AND KEEP UPDATE LOCKS clause. However, not every SQL statement necessarily supports USE AND KEEP UPDATE LOCKS; for example, if the query is being run against multiple tables in a join, USE AND KEEP UPDATE LOCKS is not supported. To avoid this problem, try either of the following:

- Modify your SQL query or reconfigure your application so that an update lock is supported
- 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.

Important file for message-driven beans

The following 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.

Client applications

Application client troubleshooting tips

This section provides some 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 UNIX platforms) 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

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.

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.

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

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.

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 Web Services Description Language (WSDL) file with `http`, `jms` and `ejb` bindings generated by the **Java2WSDL** 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 Enterprise JavaBeans (EJB) implementation. The servlet and session EJB are located in the `webservices.xml` file represented by <servlet-link> or <ejb-link> element.

The **WSDL2Java** 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 (<session>) or Java bean (<servlet-class>). 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. Then, you create the stock quote Web service, StockQuote. The **WSDL2Java** 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 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.bat` file when using a Windows operating system or the `WSDL2Java.sh` file if you are using a Linux or Unix operating system. These files are located in the *<install_root>*`/WebSphere/AppServer/bin` directory.

If you use a Windows operating system, set your proxy host and port values in the `WSDL2Java.bat` file:

```
PROXY_INFO="-Dproxy.httpHost=yourPproxyHost -Dproxy.httpPort=yourProxyPort
```

If you use a Linux or Unix operating system, set your proxy host and port values in the `WSDL2Java.sh` file:

```
PROXY_INFO="-Dproxy.httpHost=yourProxyHost -Dproxy.httpPort=yourProxyPort
```

Emitter failure error occurs when running the WSDL2Java command on a WSDL document containing a JMS-style endpoint URL

If you run the **WSDL2Java** command-line tool on a WSDL document that contains a JMS-style endpoint Web address, for example, `jms:/...`, the `urlprotocols.jar` file that contains the custom protocol handler for the JMS protocol must be in the CLASSPATH variable statement. The error **WSWS3099E: Error: Emitter failure. Invalid endpoint address in port <x> in service <y>: <jms-url-string>** is avoided by making sure the `urlprotocols.jar` file is in the CLASSPATH variable statement.

To add the `urlprotocols.jar` file to the CLASSPATH variable statement:

On Windows platforms, edit the *install_root*\bin\setupCmdLine.bat file and locate the line that sets the *WAS_CLASSPATH* environment variable. Add *%install_root%*\lib\urlprotocols.jar to the end of the line that sets the *WAS_CLASSPATH* environment variable.

On Linux and Unix platforms, 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.

Make sure to use the proper deliminator character for your platform, for example, use a semi-colon (;) for Windows platforms and a colon (:) for Linux and Unix platforms.

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 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.

Runtime migration error

If you installed a Web service application that was developed for a WebSphere Application Server version prior to Version 6.0, 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 6.0. 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. Set the value using the administrative console. Click **Servers** > **Application Servers** > ***server_name*** > ***Web Container*** > ***HTTP Transports*** > ***port_number*** > **Custom Properties** > **New**. 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.

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) 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 (serialize) a Java Date into XML code and then turn it back (deserialize) 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 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** command-line 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;
}
```

Note: 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 Web Services for J2EE

These Web services are developed and implemented based on the Web Services for Java 2 platform, Enterprise Edition (J2EE) specification. This topic discusses the following troubleshooting authentication and authorization when you are securing Web services:

- Authentication challenge or authorization failure is displayed
- Web services security enabled application fails to start
- Applications with Web services security enabled cannot interoperate between WebSphere Application Server Version 6 and Version 5.0.2

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 Java 2 Platform, Enterprise Edition (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.

Tracing Web services

You can trace the Web services runtime components, including an unmanaged client, managed client and a server application. The procedure entry and exit, as well as the processing actions are traceable in the runtime components. You can also trace user-defined exceptions, as well as SOAP messages that use Java Message Service (JMS) or HTTP to request Web services.

The `com.ibm.ws.webservices.engine.*=all=enabled` specification traces the Web services runtime only. See the step 4 for settings that you can use to trace user-defined exceptions and SOAP messages or review Tracing SOAP messages with tcpmon to learn about tracing SOAP messages with the tcpmon process.

The following tasks describe how you can enable trace for Web services:

1. Enable trace for a Web services unmanaged client.
   a. Create a trace properties file by copying the *%install_root*`\properties\TraceSettings.properties` file to the same directory as your client application Java archive (JAR) file.
   b. Edit the properties file and change the value of traceFileName to output the trace data. For example, `traceFileName=c:\\temp\\myAppClient.trc`.
   c. Edit the properties file to remove `com.ibm.ejs.ras.*=all=enabled` and add `com.ibm.ws.webservices.engine.*=all=enabled`.
   d. Add the `-DtraceSettingsFile=<`*trace_properties_file*`>` option to the Java command line that is used to run the client, where *trace_properties_file* represents the name of the properties file that you created in the substeps a through c. For example, `java -DtraceSettingsFile=TraceSettings.properties myApp.myAppMainClass`.
2. Enable trace for a Web services-managed client by invoking the **launchClient** command-line tool with the following options:

   -CCtrace=com.ibm.ws.webservices.engine.*=all=enabled

   -CCtracefile=traceFileName For example:

   %install_root%`\bin\launchClient MyAppClient.ear`

   `-CCtrace=com.ibm.ws.webservices.engine.*=all=enabled -CCtracefile=myAppClient.trc`

   See launchClient tool for more information.
3. Enable trace for a Web Services for J2EE server application.
   a. Start WebSphere Application Server.
   b. Open the administrative console.
   c. Click **Servers** >**Application Servers** > *server*.
   d. Click **Diagnostic Trace Service**.
   e. In the Trace Specification field, delete the text `*=all=enabled` and add `com.ibm.ws.webservices.engine.*=all=enabled`.
   f. Click **Save** and **Apply**.
4. Enable trace for either SOAP messages or user-defined exceptions, or both. The following trace specifications are used to trace SOAP messages:
   - `com.ibm.ws.webservices.trace.MessageTrace=all`

This specification traces the contents of a SOAP message, including the binary attachment data.

When the context-type of the SOAP message is not text/xml, the message probably contains attachments. In this case, the message is displayed in the trace file in the hex dump format. The following example illustrates a line in the hex dump format for non-text SOAP messages:

```
0000: 0D 0A 2D 2D 2D 2D 2D 2D - 3D 5F 50 61 72 74 5F 36 ..------=_Part_6
```

– 16 bytes of the message are displayed in each trace file line
– The first four digits are a hex number whose value is the byte offset into the SOAP message of the first byte on the line.
– The next 16 two-digit hex numbers are the contents of each of the consecutive bytes in the message.
– The ASCII representation of the bytes are displayed in the last 16 characters of the line, with unprintable characters represented by a period.

- `*=off:com.ibm.ws.webservices.*=all`

  You can trace all Web services information, including the SOAP messages and user-defined exceptions, with this setting.

The following steps explain how to trace user-defined exceptions:

a. Enable logging of user-defined exceptions by specifying the `com.ibm.ws.webservices.trace.UserExceptionTrace=all` trace string

   The user-defined exceptions are not logged by default. A user-defined exception is an exception that is defined in the Web Services Description Language (WSDL) file for an operation.

   A user-defined exception often indicates an error-free condition. For example, a user-defined exception often indicates an error-free condition. For example, the user-defined `OverdrawnException` exception, can occur for the service endpoint implementation of the makeWithdrawl() method. This exception indicates an expected condition and does not indicate an error in the service endpoint implementation. Because these types of exceptions can occur during normal processing, they are not logged by default.

   Note: When a user-defined exception is logged, the information is sent to the `trace.log` file and not to the `SystemOut.log` file.

   You can also use the following trace strings to enable tracing for user-defined exceptions, as well as other trace points:

   - com.ibm.ws.webservices.*=all

     Turns on all Web services runtime trace logs.

   - com.ibm.ws.webservices.trace.*=all

     Turns on MessageTrace and UserExceptionTrace.

You have enabled trace for the unmanaged clients, managed clients and the server applications. Depending on the trace string specification, the trace can include runtime components, user-defined exceptions and SOAP messages.

Analyze the message data.

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. See Tracing Web services for more information about these other trace tools.

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.
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.

Frequently asked questions about Web services

This topic presents frequently asked questions about Web services that are developed and implemented based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification.

- What IBM development tools work with Web Services that are developed based on the Web Services for J2EE specification?
- Is Web Services for J2EE technology part of the J2EE specification?
- What is the relationship between the Web Services for J2EE specification and the Web Service Invocation Framework (WSIF)?
- What is the relationship between Apache SOAP 2.3 and the Web Services for J2EE specification?
- What standards does the Web Services for J2EE component of WebSphere Application Server Version 6.0 support?
- Does the Web Services for J2EE technology interoperate with other SOAP implementations, like .NET?
- Can I use a JavaBeans component to implement a Web service using SOAP Java Message Service (JMS) invocation?
- Does the SOAP and JMS support interoperate with other vendors?
- How does two-way messaging with a SOAP and JMS implementation work? Can it support multiple clients making simultaneous requests?

What IBM development tools work with Web Services that are developed based on the Web Services for J2EE specification?

The assembly tools, Application Server Toolkit (AST) and Rational Web Developer, provide a graphical interface for developing code artifacts, assembling the code artifacts into various archives (modules) and configuring related Java 2 Platform, Enterprise Edition (J2EE) Version 1.2, 1.3 or 1.4 compliant deployment descriptors.

Is Web Services for J2EE technology part of the J2EE specification?

WebSphere Application Server Version 6.0 is based on J2EE 1.4. For WebSphere Application Server Version 5.0.2 and Version 5.1.x, the Web Services for J2EE Version 1.0 specification is an addition to J2EE 1.3. The J2EE specification 1.4 requires support for Web Services for J2EE Version 1.1. Minor differences exist between the J2EE 1.3 Version (JSR-109 Version 1.0) and the J2EE 1.4 Version (JSR-109 Version 1.1).

What is the relationship between the Web Services for J2EE specification and the Web Service Invocation Framework (WSIF)?

Web Services for J2EE and WSIF represent two different programming models for accessing Web services. Web Services for J2EE is standard, Java-centric, and more statically bound to Web Services Description Language (WSDL) documents because of the use of generated stubs. WSIF directly models WSDL. WSIF is more suitable when dynamically interpreting WSDL. WebSphere Application Server Version 6.0 leverages both technologies to achieve dynamic, high performing standards-based Web services implementations.

What is the relationship between Apache SOAP 2.3 and the Web Services for J2EE specification?

Apache SOAP ships with WebSphere Application Server Versions 4.0 and 5.0. It continues to co-exist with Web services that are based on the Web Services for J2EE technology. Apache SOAP is a proprietary API and applications written for it are not portable to other SOAP implementations. Applications written according to the Web Services for J2EE are portable to any vendor implementation that supports this specification.

What standards does the Web Services for J2EE component of WebSphere Application Server Version 6.0 support?

The following standards are supported by the Web Services for J2EE component of WebSphere Application Server Version 6.0:
- SOAP Version 1.1
- Web Services Description Language (WSDL) Version 1.1
- Web Services for J2EE Version 1.1
- Java API for XML-Based RPC (JAX-RPC) Version 1.1
- SOAP with attachments API for Java (SAAJ) Version 1.2

Does the Web Services for J2EE technology interoperate with other SOAP implementations, like .NET?

WebSphere Application Server Version 6.0 supports Web services that are consistent with the WS-I Basic Profile 1.0, and should interoperate with any other vendor conforming to this specification.

Can I use a JavaBeans component to implement a Web service using SOAP Java Message Service (JMS) invocation?

The SOAP and JMS support uses message-driven beans (MDB) to implement the JMS endpoint. You can use MDBs in the Enterprise JavaBeans (EJB) container and delegated to an enterprise bean. If you want to use a JavaBeans instead of an enterprise bean to implement the service endpoint, you must create a *facade* enterprise bean that delegates to the JavaBeans implementation.

Does the SOAP and JMS support interoperate with other vendors?

No. Currently no specification exists for SOAP and JMS invocationss, therefore each vendor chooses an implementation technique.

How does two-way messaging with a SOAP and JMS implementation work? Can it support multiple clients making simultaneous requests?

Before a client issues a two-way request, it creates a temporary JMS queue to receive the response. This temporary queue is specified as the replyTo destination that is in the outgoing JMS request message. After the server processes the request, it directs the response to the replyTo destination specified in the request message. The client deletes the temporary queue after the response is received. The server can handle simultaneous requests from multiple clients because each incoming request message contains the destination to which the reply is sent.

Troubleshooting the Web Services Invocation Framework

For information on 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 WSDL can cause numerous problems. To help ensure that your WSDL is valid, use a tool such as Rational Application Developer to create your service.

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 Rational Application Developer 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, see the Address Book Sample.

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, see Enabling security for WSIF.

WSIF setup
: Check that the `wsif.properties` file is correctly set up. For more information, see Maintaining the WSIF properties file.

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.

Here is a list of common errors, and information on their probable causes:

- **"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, see Enabling security for WSIF.

- **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.

- **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. To enable interoperation, modify either your Web service or the WSIF default SOAP provider as described in WSIF SOAP provider: working with legacy applications.

Trace and logging for WSIF

If you want to enable trace for the Web Services Invocation Framework (WSIF) API within WebSphere Application Server, and have trace, stdout and stderr for the application server written to a well-known location, see Enabling trace.

WSIF offers trace points at the opening and closing of ports, the invocation of services, and the responses from services.

To trace the WSIF API, you need to specify the following trace string:

```
wsif=all=enabled
```

WSIF also includes a `SimpleLog` utility through which you can run trace when using WSIF outside of WebSphere Application Server. To enable this utility, 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` uitility 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.

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

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 IBM WebSphere UDDI Registry is running, it might issue messages to report events or errors. You can use these messages, described in Messages as your first aid to problem determination. If you need more details about the causes of a problem, you can turn on tracing for UDDI, as described in:

- Turning on UDDI Registry trace

Some errors that you might encounter when setting up or using the UDDI Registry, and their causes, are described in Common Causes of Errors in the UDDI Registry

Turning on UDDI Registry trace

To enable trace, specify `com.ibm.uddi.*` as the trace specifier:

```
'com.ibm.uddi.*=all=enabled'
```

This enables all types of trace for the UDDI Registry. Please refer to ″Enabling trace″ elsewhere in this Information Center for more information about using the administrative console to enable or disable trace.

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.
- On Unix and Linux platforms, if using DB2, run the db2profile script before issuing the startServer.sh server1 command. This script is located within the DB2 instance home directory under sqllib and is invoked by typing:

  ```
  . /home/db2inst1/sqllib/db2profile
  ```

  Note: In the above example, notice that the '.' is followed by a single space character.

  Note: On Unix and Linux platforms the DB2 user **must** have a db2profile at $HOME/sqllib/db2profile
- 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 run the uddiDeploy.jacl with the default option, and you have not first deleted the UDDI Cloudscape database (from the databases subdirectory of your server profile) the UDDI database will not be recreated, and you will get an error message. This error can be ignored if you did not intend to replace the Cloudscape database.
- If attempting to use a remote DB2 database and you 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. If, however, the remote system is using DHCP, the two systems must be aware of each others subnet mask.

  For Windows, the subnet mask can be found by starting a Command Prompt and entering *"ipconfig"* on the remote system. On the host system, the WINS might need to be edited to add the remote subnet mask. To do this on Windows go to the following commands:
  1. START => Network and Dial-up Connections => Local Area Network Connection 2 => Internet Protocol (TCP/IP) and click on Properties
  2. Click on *"Advanced"*.
  3. Click on the WINS tab and add the new subnet mask
  4. Move the new subnet mask to the top of the list by highlighting it and pressing the *"Up"* arrow until it is the top of the list of WINS addresses

  On Unix platforms, you can use *ifconfig* to determine the 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 WebSphere 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 IBM WebSphere UDDI Registry component to IBM, supply the following information:
1. A detailed description of the problem.

2. The build date and time of the version you are using. This can be obtained as follows:
   - In the WAS_HOME/profiles/<profile_name>/installedApps/<cell_name> subdirectory of the WebSphere installation location, you will find a subdirectory called UDDI_Registry.<nodename>.<servername>.ear, where <nodename> is the name of the node into which the UDDI Registry application is installed, and <servername> 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 ″getUDDIMessageLogger″ and ″UDDI Build :″ followed by the build date and time, and the build system. Also include this information.
3. Any relevant log files and trace files.
   - If the problem occurred while setting up and installing the UDDI Registry application 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 section 'Setting up and deploying the UDDI Registry'.) The log file is written to the directory from which you ran the setup script.
   - If the problem occurred while removing the UDDI Registry application using the remove script, uddiRemove.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 section 'Removing a user customized UDDI Registry node' or 'Removing a default UDDI Registry node'.) The log file is written to the directory from which you ran the remove script.
   - 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=enabled″.
   - 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 executing the command `versioninfo (Windows) or versioninfo.sh (Unix platforms)` on the application server node and directing the output to a log file.
4. If appropriate, any application code that you are using and the output produced by the application code.
5. 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.

Removing a user customized UDDI Registry node:

To completely remove the UDDI Registry application from the target application server in the deployment manager cell, run the *wsadmin* (*wsadmin.sh* on Unix Platforms) script *uddiRemove.jacl*, which is located in the <WAS_install_dir>\bin directory of the deployment manager install tree.

At a command prompt enter:

```
wsadmin -f uddiRemove.jacl
            servername
            nodename
            > uddiremove.log
```

Where *servername* and *nodename* are the server and node where you have deployed the UDDI Registry application. By default output will go to the screen, but, optionally, you can specify '> uddiremove.log' to direct output to a log file. For example,

```
wsadmin -f uddiRemove.jacl server1 myriad
```

will remove the UDDI Registry application and related files from server server1 running in node myriad, and will send any messages to the screen.

In the case of a Network Deployment configuration, the default option cannot be used in the deployment manager.

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 non pooled 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);
  }
   }

}
```

Database exceptions resulting from foreign key conflicts, or deadlock when entity beans are configured for optimistic concurrency control

Exceptions resulting from foreign key conflicts, which signify violations of database referential integrity

A database *referential integrity* (RI) policy prescribes rules for how data is written to and deleted from the database tables to maintain relational consistency. Run-time requirements for managing bean persistence, however, can cause an EJB application to violate RI rules, which can cause database exceptions.

Your EJB application is violating database RI if you see an exception message in your WebSphere Application Server trace or log file that is similar to one of the following messages (which were produced in an environment running DB2):

```
The insert or update value of the FOREIGN KEY table1.name_of_foreign_key_constraint
is not equal to any value of the parent key of the parent table.
```

or

```
A parent row cannot be deleted because the relationship table1.name_of_foreign_key_constraint
is not equal to any value of the parent key of the parent table.
```

To prevent these exceptions, you must designate the order in which entity beans update relational database tables by defining sequence groups for the beans.

Exceptions resulting from deadlock caused by optimistic concurrency control schemes

Additionally, sequence grouping can minimize transaction rollback exceptions for entity beans that are configured for optimistic concurrency control. Optimistic concurrency control dictates that database locks be held for minimal amounts of time, so that a maximum number of transactions consistently have access to the data. In such a highly available database, concurrent transactions can attempt to lock the same table row and create deadlock. The resulting exceptions can generate messages similar to the following (which was produced in an environment running DB2):

```
Unsuccessful execution caused by deadlock or timeout.
```

Use the sequence grouping feature to order bean persistence so that database deadlock is less likely to occur.

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()");
      }
   }
}
```

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. A list that contains detailed descriptions for all of these providers follows the table.

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	
	DB2 legacy CLI-based Type 2 JDBC Provider	One phase only	
	DB2 legacy CLI-based Type 2 JDBC Provider (XA)	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	Recommended for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX
	DB2 UDB for iSeries (Toolbox XA)	One and two phase	Recommended for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX
	DB2 legacy CLI-based Type 2 JDBC Provider	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)	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)	One and two phase	*Only* for use with WebSphere Application Server **run on z/OS**
	DB2 Universal JDBC Provider	One phase only	*Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX
	DB2 Universal JDBC Provider (XA)	One and two phase	*Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX
	DB2 legacy CLI-based Type 2 JDBC Provider	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)	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)
Cloudscape	Cloudscape JDBC Provider	One phase only	**-** Not for use in clustering environment: Cloudscape is accessible from a single JVM only **-** Does not support Version 4 data sources
	Cloudscape JDBC Provider (XA)	One and two phase	**-** Not for use in clustering environment: Cloudscape is accessible from a single JVM only **-** Does not support Version 4 data sources
	Cloudscape Network Server using Universal JDBC driver	One phase only	Does not support Version 4 data sources
Informix	Informix JDBC Driver	One phase only	
	Informix JDBC Driver (XA)	One and two phase	
Sybase	Sybase JDBC Driver	One phase only	
	Sybase JDBC Driver (XA)	One and two phase	
Oracle	Oracle JDBC Driver	One phase only	
	Oracle JDBC Driver (XA)	One and two phase	

Database type	JDBC Provider	Transaction support	Version and other considerations
MS SQL Server	DataDirect ConnectJDBC type 4 driver for MS SQL Server	One phase only	Only for use with the corresponding driver from DataDirect Technologies
	DataDirect ConnectJDBC type 4 driver for MS SQL Server (XA)	One and two phase	Only for use with the corresponding driver from DataDirect Technologies
	WebSphere embedded ConnectJDBC driver for MS SQL Server	One phase only	- Not available for Application Server on z/OS - Cannot be used outside of WebSphere Application Server environment
	WebSphere embedded ConnectJDBC driver for MS SQL Server (XA)	One and two phase	- Not available for Application Server on z/OS - Cannot be used outside of WebSphere Application Server environment

Detailed JDBC provider list

The following list contains a description of every JDBC provider that can be defined for use with WebSphere Application Server Version 6.x. It also shows the supported data source classes and their required properties. 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 properties list. Rather, inclusion in the list means that a value is typically required for that field.

Use these links to find your provider information:
- DB2 on Windows, UNIX, or workstation-based LINUX
- DB2 UDB for iSeries
- DB2 on z/OS, connecting to Application Server on Windows, UNIX, or workstation-based LINUX
- Cloudscape
- Informix
- Sybase
- Oracle
- MS SQL Server

DB2 on Windows, UNIX, or workstation-based LINUX

1. **DB2 Universal JDBC Provider**

   The DB2 Universal JDBC Driver is an architecture-neutral JDBC driver for distributed and local DB2 access. Because the Universal Driver architecture is independent of any particular JDBC driver connectivity or target platform, it allows both Java connectivity (Type 4) or Java Native Interface (JNI) based connectivity (Type 2) in a single driver instance to DB2.

   This driver only supports one phase transactions. This JDBC driver allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access.

   The DB2 Universal JDBC Driver Provider supports one phase data source:

   com.ibm.db2.jcc.DB2ConnectionPoolDataSource

   Requires JDBC driver files:
   - **db2jcc.jar** After you install DB2, you can find this jar file in the DB2 *java* directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must

also set the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable to point to the **db2jcc.jar** file. See the Cloudscape section for more information on the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable.

Note: To find out the version of the universal driver you are using, issue this DB2 command:

```
java com.ibm.db2.jcc.DB2Jcc -version
```

The output for the above example is:

```
IBM DB2 JDBC Universal Driver Architecture 2.2.xx
```

- **db2jcc_license_cu.jar** This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this jar file or the next one to gain access to the database. This jar file ships with WebSphere Application Server in a directory defined by **${UNIVERSAL_JDBC_DRIVER_PATH}**environment variable.
- **db2jcc_license_cisuz.jar** This is the DB2 Universal JDBC driver license file that allows access to the following databases:
  - DB2 Universal
  - DB2 for iSeries
  - DB2 for z/OS
  - SQLDS

  The **db2jcc_license_cisuz.jar** does not ship with Websphere Application Server and should be located in the same directory as the db2jcc.jar file, so that the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** points to both.

```
The classpath for this provider is set as follows:

   <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar </classpath>
   <classpath>${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar</classpath>
   <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar</classpath>
```

Note: The license jar files are independent of each other; therefore, order does not matter.

Requires **DataStoreHelper** class:

```
com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
```

Requires a valid authentication alias.

Requires properties:

- **databaseName** This is an actual database name if the **driverType** is set to **4**, or a locally cataloged database name if the **driverType** is set to **2**.
- **driverType** The JDBC connectivity type of a data source. *There are two permitted values: 2 and 4*. If you want to use Universal JDBC Type 2 driver, set this value to 2. If you want to use Universal JDBC Type 4 driver, set this value to 4.
- **serverName** The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **portNumber** The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.

2. **DB2 Universal JDBC Provider (XA)**

   The DB2 Universal JDBC Provider (XA) is an architecture-neutral JDBC provider for distributed and local DB2 access. Whether you use this provider for Java connectivity or Java Native Interface (JNI) based connectivity depends on the version of DB2 you are running. Application Server Version 6.0 minimally requires DB2 8.1 Fix Pack 6. This version of DB2 only supports XA connectivity over the Java Native Interface (JNI) based connectivity (Type 2) driver. In order to use XA connectivity with the Type 4 driver, DB2 8.1 Fix Pack 7 or higher is required.

   The DB2 Universal JDBC Driver (XA) supports two phase transactions and the more advanced data source option offered by Application Server (as opposed to the other option, Version 4 data sources). This driver also allows applications to use both JDBC and SQLJ access.

The DB2 Universal JDBC Driver Provider supports the two phase data source:

com.ibm.db2.jcc.DB2XADataSource

Requires JDBC driver files:

- **db2jcc.jar** After you install DB2, you can find this `.jar` file in the DB2 java directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must also set the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** environment variable to point to the `db2jcc.jar` file. See the Cloudscape section for more information on the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** environment variable.

  Note: To find the level of universal driver you are using, issue the following DB2 command:

  ```
  java com.ibm.db2.jcc.DB2Jcc -version
  ```

  example output of the above:

  ```
  IBM DB2 JDBC Universal Driver Architecture 2.2.xx
  ```

- **db2jcc_license_cu.jar** This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this jar file or the next one to gain access to the database. This jar file ships with WebSphere Application Server in the *WAS_HOME/universalDriver/lib* directory.
- **db2jcc_license_cisuz.jar** This is the DB2 Universal JDBC driver license file that allows access to the following databases:
  - DB2 Universal
  - DB2 for iSeries
  - DB2 for z/OS
  - SQLDS

  You must use the right license jar file to access a specific database backend.

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **databaseName** This is an actual database name if the **driverType** is set to **4**, or a locally cataloged database name if the **driverType** is set to **2**.
- **driverType** The JDBC connectivity type of a data source. *There are two permitted values: 2 and 4*. If you want to use Universal JDBC Type 2 XA driver, set this value to 2. If you want to use Universal JDBC Type 4 XA driver (which requires DB2 8.1 Fix Pack 7 or higher), set this value to 4.
- **serverName** The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **portNumber** The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.

3. **DB2 legacy CLI-based Type 2 JDBC Driver**

   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.

   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 if Application Server is running on the same machine as the database. Otherwise, connectivity through this driver does require an alias.

   Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

4. **DB2 legacy CLI-based Type 2 JDBC Driver (XA)**

   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.

   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** (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 if Application Server is running on the same machine as the database. Otherwise, connectivity through this driver does require an alias.

   Requires properties:
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

For more information on DB2, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/.

For information on configuring WebSphere Application Server for DB2 access, see the Configuring DB2 article.

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**

   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)**

   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** (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*.

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/

DB2 on z/OS, connecting to Application Server on Windows, UNIX, or workstation-based LINUX

1. **DB2 Universal JDBC Provider**

   The DB2 Universal JDBC Driver is an architecture-neutral JDBC driver for distributed and local DB2 access. Because the Universal Driver architecture is independent of any particular JDBC driver connectivity or target platform, it allows both Java connectivity (Type 4) or Java Native Interface (JNI) based connectivity (Type 2) in a single driver instance to DB2. Starting with WebSphere Application Server Version 5.0.2, the product now supports both Type 2 and Type 4 JDBC drivers. To use the Type 4 driver, you must install DB2 Version 8.1 or a later version. To use the Type 2 driver, you must install DB2 Version 8.1 Fix Pack 2 or a later version.

   This driver only supports one phase transactions. This JDBC driver allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access.

   The DB2 Universal JDBC Driver Provider supports one phase data source:

   com.ibm.db2.jcc.DB2ConnectionPoolDataSource

   Requires JDBC driver files:

   - **db2jcc.jar** After you install DB2, you can find this jar file in the DB2 *java* directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must also set the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable to point to the **db2jcc.jar** file. See the Cloudscape section for more information on the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable.

     Note: To find out the version of the universal driver you are using, issue this DB2 command:

     ```
     java com.ibm.db2.jcc.DB2Jcc -version
     ```

     The output for the above example is:

     ```
     IBM DB2 JDBC Universal Driver Architecture 2.2.xx
     ```

   - **db2jcc_license_cu.jar** This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this jar file or the next one to gain access to the database. This jar file ships with WebSphere Application Server in a directory defined by **${UNIVERSAL_JDBC_DRIVER_PATH}**environment variable.
   - **db2jcc_license_cisuz.jar** This is the DB2 Universal JDBC driver license file that allows access to the following databases:
     – DB2 Universal
     – DB2 for iSeries
     – DB2 for z/OS
     – SQLDS

     The **db2jcc_license_cisuz.jar** does not ship with Websphere Application Server and should be located in the same directory as the db2jcc.jar file, so that the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** points to both.

   ```
   The classpath for this provider is set as follows:

      <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar </classpath>
      <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar</classpath>
      <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar</classpath>
   ```

   Note: The license jar files are independent of each other; therefore, order does not matter.

   Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **databaseName** This is an actual database name if the **driverType** is set to **4**, or a locally cataloged database name if the **driverType** is set to **2**.
- **driverType** The JDBC connectivity type of a data source. *There are two permitted values: 2 and 4*. If you want to use Universal JDBC Type 2 driver, set this value to 2. If you want to use Universal JDBC Type 4 driver, set this value to 4.
- **serverName** The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **portNumber** The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.

2. **DB2 Universal JDBC Provider (XA)**

   The DB2 Universal JDBC Driver (XA) is an architecture-neutral JDBC driver for distributed and local DB2 access. In WebSphere Application Server Version 5.0.2, this driver only supports Java Native Interface (JNI) based connectivity (Type 2) in a single driver instance to DB2. To use this driver, you must install DB2 Version 8.1 Fix Pack 2 or a later version. This driver supports two phase transactions and the WebSphere Application Server Version 5.0 data source. This driver allows applications to use both JDBC and SQLJ access.

   The DB2 Universal JDBC Driver Provider supports the two phase data source:

   `com.ibm.db2.jcc.DB2XADataSource`

   Requires JDBC driver files:

   - **`db2jcc.jar`** After you install DB2, you can find this `.jar` file in the DB2 java directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must also set the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** environment variable to point to the `db2jcc.jar` file. See the Cloudscape section for more information on the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** environment variable.

     Note: To find the level of universal driver you are using, issue the following DB2 command:

     ```
     java com.ibm.db2.jcc.DB2Jcc -version
     ```

     example output of the above:

     ```
     IBM DB2 JDBC Universal Driver Architecture 2.2.xx
     ```

   - **`db2jcc_license_cu.jar`** This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this jar file or the next one to gain access to the database. This jar file ships with WebSphere Application Server in the *WAS_HOME/universalDriver/lib* directory.
   - **`db2jcc_license_cisuz.jar`** This is the DB2 Universal JDBC driver license file that allows access to the following databases:
     - DB2 Universal
     - DB2 for iSeries
     - DB2 for z/OS
     - SQLDS

     You must use the right license jar file to access a specific database backend.

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **databaseName** This is a locally cataloged database name.
   - **driverType** This is the JDBC connectivity type of a data source. *If you are running a version of DB2 prior to DB2 V8.1 FP6, you are restricted to using only the type 2 driver.*

- **serverName** The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **portNumber** The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.

3. **DB2 legacy CLI-based Type 2 JDBC Driver**

   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. For use with Application Server on Windows, UNIX, or workstation-based LINUX, this provider requires DB2 Connect (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*.

4. **DB2 legacy CLI-based Type 2 JDBC Driver (XA)**

   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. For use with Application Server on Windows, UNIX, or workstation-based LINUX, this provider requires DB2 Connect (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** (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*.

For more information on DB2 for z/OS, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/.

Cloudscape

1. **Cloudscape JDBC Provider**

   The Cloudscape JDBC Provider provides the JDBC access to the Cloudscape database. This Cloudscape JDBC driver used the embedded framework. You cannot use any Version 4.0 data sources with Cloudscape.

   Cloudscape JDBC Provider supports one phase data source:

   com.ibm.db2j.jdbc.DB2jConnectionPoolDataSource

   Requires JDBC driver files: **db2j.jar.**

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper`

   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./cloudscape` (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. **Cloudscape JDBC Provider (XA)**

   The Cloudscape JDBC Provider (XA) provides the XA-compliant JDBC access to the Cloudscape database. This Cloudscape JDBC driver uses the embedded framework. You cannot use any Version 4.0 data sources with Cloudscape.

   Cloudscape JDBC Provider (XA) supports two phase data source:

   com.ibm.db2j.jdbc.DB2jXADataSource

   Requires JDBC driver files: **db2j.jar**

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper`

   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./cloudscape` (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. **Cloudscape Network Server using Universal JDBC driver**

   This Cloudscape 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.

   Cloudscape uses the DB2 Universal Driver when using the Network Server. It supports one phase data source:

   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 Cloudscape network server is set to be:

     ```
     <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar</classpath>

     <classpath>${CLOUDSCAPE_JDBC_DRIVER_PATH}/db2j.jar</classpath>

     <classpath>${CLOUDSCAPE51_JDBC_DRIVER_PATH}/db2j.jar</classpath>

     <classpath>${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.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 when you set up your DB2 datasource. This is instead of hard coding the path of the db2jcc.jar for DB2 datasources.
   - **db2jcc_license_cu.jar** This file is the DB2 Universal JDBC license file that provides access to the Cloudscape 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 defined to the location in Websphere Application Server where the license jar file above 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 vesions 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. See the DB2 section for more information on the **DB2UNIVERSAL_JDBC_DRIVER_PATH** path variable.

Note: Cloudscape requires only db2jcc_license_c.jar; however, WebSphere Application Server uses db2jcc_license_cu.jar because this works for both DB2 UDB and Cloudscape.

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.CloudscapeNetworkServerDataStoreHelper`

Note: The administrative console incorrectly lists the DB2UniversalDataStoreHelper as the default value for the **DataStoreHelper** class. You must change the default value to `com.ibm.websphere.rsadapter.CloudscapeNetworkServerDataStoreHelper`. Also change the custom properties, using the instructions in the customer property section.

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./cloudscape` (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.

See the Cloudscape setup instructions for more information on configuring the Cloudscape Network Server.

For more information on IBM Cloudscape, visit the Cloudscape Web site at:
http://www.ibm.com/software/data/cloudscape/

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*.
- **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

   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*.
   - **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*.

For more information on Informix, visit the Informix Web site at: http://www.ibm.com/software/data/informix/

Sybase

1. **Sybase JDBC Driver**

   The Sybase JDBC Driver is a Type 4 JDBC driver that provides JDBC access to the Sybase database.

   Sybase JDBC Driver supports one phase data source:

   com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource

   Requires JDBC driver files: **jconn2.jar**.

   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 JDBC Driver (XA)**

The Sybase JDBC Driver (XA) is a Type 4 JDBC driver that provides XA-compliant JDBC access to the Sybase database.

Sybase JDBC Driver (XA) supports two phase data source:

com.sybase.jdbc2.jdbc.SybXADataSource

Requires JDBC driver files: **jconn2.jar**.

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`

   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

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 database. 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 database. 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. **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 database. 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. **WebSphere embedded ConnectJDBC driver for MS SQL Server (XA)**

   WebSphere embedded ConnectJDBC driver for MS SQL Server (XA) is a Type 4 JDBC driver which provides XA-compliant JDBC access to the MS SQL Server database. This JDBC driver ships with WebSphere Application Server. Use this provider with the Connect JDBC driver embedded in WebSphere. Do not use it 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*.

   Whether you need to support one or two phase transactions with the WebSphere embedded Connect JDBC XA driver, you must install Stored Procedures for the Java Transaction API (JTA) on your machine that runs Microsoft SQL. The WebSphere Application Server installation disks provide a base level of Stored Procedures for JTA. (You can find the most current version of the software on the WebSphere Application Server-embedded DataDirect Technologies product update Web page.) 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 theinstjdbc.sql script. (The script can be run by the MS SQL Server Query Analyzer or the ISQL utility).

You can download the latest patches and upgrades to the WebSphere embedded Connect JDBC driver from the following FTP site:

`ftp://ftp.software.ibm.com/software/websphere/info/tools/DataDirect/datadirect.htm`

5. **DataDirect SequeLink type 3 JDBC driver for MS SQL Server** -- Deprecated

   Because this JDBC provider is deprecated in WebSphere Application Server Version 6.0, it is no longer an available option in the administrative console. In its place, use one of the Connect JDBC providers, which are described previously in this section.

   DataDirect SequeLink type 3 JDBC driver for MS SQL Server is a type 3 JDBC driver that provides JDBC access to MS SQL Server via SequeLink server.

   This JDBC provider supports this 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*.

6. **DataDirect SequeLink type 3 JDBC driver for MS SQL Server (XA)** -- Deprecated

   Because this JDBC provider is deprecated in WebSphere Application Server Version 6.0, it is no longer an available option in the administrative console. In its place, use one of the Connect JDBC providers, which are described previously in this section.

   DataDirect SequeLink type 3 JDBC driver for MS SQL Server (XA) is a type 3 JDBC driver that provides XA-compliant JDBC access to MS SQL Server via the SequeLink server.

   This JDBC provider supports this 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*.

Both of the WebSphere-embedded SequeLink JDBC drivers require 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. (Only install SequeLink Server from the WebSphere Application Server CD if you are using the SequeLink JDBC driver embedded in WebSphere. Otherwise, install a copy of SequeLink Server purchased from DataDirect Technologies.)

From the following FTP site, you can download the latest patches and upgrades for the version of SequeLink Server that is used with the WebSphere-embedded SequeLink JDBC driver:

`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/`

7. **Microsoft JDBC driver for MSSQLServer 2000** -- Deprecated

   Because this JDBC provider is deprecated in WebSphere Application Server Version 6.0, it is no longer an available option in the administrative console. In its place, use one of the Connect JDBC providers, which are described previously in this section.

   Microsoft JDBC driver for MSSQLServer 2000 is a type 4 JDBC driver that provides JDBC access to the MS SQL Server database.

   This JDBC provider supports this 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*.

8. **Microsoft JDBC driver for MSSQLServer 2000 (XA)** -- Deprecated

   Because this JDBC provider is deprecated in WebSphere Application Server Version 6.0, it is no longer an available option in the administrative console. In its place, use one of the Connect JDBC providers, which are described previously in this section.

   Microsoft JDBC driver for MSSQLServer 2000 (XA) is a type 4 JDBC driver that provides XA-compliant JDBC access to the MS SQL Server database.

   This JDBC provider supports this 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*.

For more information on the Microsoft JDBC driver, visit the Microsoft Web site at:

`http://www.microsoft.com/sql`

Messaging resources

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.
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 221.
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, see Using messaging.
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.

Tips for troubleshooting WebSphere Messaging

This topic provides a set of tips to help you troubleshoot problems with WebSphere Messaging.

- An MDB listener fails to start

- Problems running JMS applications with security enabled
- Queue manager fails to stop on Redhat Linux
- Application server fails to start in zh_TW.EUC locale on Solaris

For messaging problems specific to WebSphere Application Server version 5 nodes, see the version 5 information center and the Application Servers support web site; for example: 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`, then the application needs to supply valid credentials.
- If the authentication mechanism is set to `Container`, then you need to 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, then the JMS Bindings authentication by WebSphere MQ throws 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 Asynchronous messaging - security considerations.

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.

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.
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.

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.

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.

Security

Troubleshooting authorization providers

This article describes the issues you might encounter using a Java 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.

Using Tivoli Access Manager as a Java 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.

Using an external provider for Java Contract for Containers authorization

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 are having problems configuring JACC, check the following:

- Ensure that the parameters are correct. For example, there should not be a number after TAM_Policy_server_hostname:7135, but there should 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" appears, 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 been started.
- Ensure that the password for sec_master is correct.
- Check the SystemOut.log and search for the string `AMAS` 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 has been configured, check the following:

- Ensure that the WebSphere Application Server and Tivoli Access Manager use the same Lightweight Directory Access Protocol (LDAP) server.
- If the message "Policy Director Authentication failed″ appears, ensure that the:
  - WebSphere Application Server LDAP serverID is the same as the "Administrator user" in the Tivoli Access Manager JACC configuration panel.
  - Tivoli Access Manager Administrator distinguished name (DN) is correct.
  - Password of the Tivoli Access Manager administrator has not expired and is valid.
  - 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) appears, do the following:
  1. Go to $WAS_HOME/profiles/profile_name/etc/tam.
  2. Change *xxxx* to an available port number in amwas.commomconfig.properties, and amwas*cellName_dmgr.properties if dmgr failed to start. If Node failed to start, change *xxx* to an available port number in amwas*cellName_nodeName_.properties. If appSever failed to start, change *xxxx* in Amwas*cellname_nodeName_serverName.properties.

The application might not deploy properly

When you click **Save**, the policy and role information is propagated to the Tivoli Access Manager policy. It 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 have configured 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 has been configured, do the following:

- 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:

  ```
  $WAS_INSTALL/java/jre/PdPerm.properties
  $WAS_INSTALL/java/jre/PdPerm.ks
  $WAS_INSTALL/profiles/profile_name/etc/tam/*
  ```

- Use a utility to clear the security configuration and return the system to the state it was in before Tivoli Access Manager JACC was configured. 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 Tivoli Access Manager JACC provider. Backup security.xml before running this utility.

  Enter the following commands:

  ```
  $WAS_HOME/java/jre/bin/java -classpath
   "$WAS_HOME/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 are attempting 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."
```

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 password
   ```

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 command propagatePolicyToJACCProvider:

```
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, which 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 task option MapRolesToUsers becomes invalid when Tivoli Access Manager is used as the authorization server. To correct the error, change MapRolesToUsers to TAMMapRolesToUsers.

Troubleshooting security configurations

Refer to Security components troubleshooting tips for instructions on how to troubleshoot errors related to security.

The following topics explain how to troubleshoot specific problems related to configuring and enabling security configurations:

- Errors when configuring or enabling security
- Errors or access problems after enabling security

- Errors trying to enable or configure Secure Socket Layer (SSL) encrypted access
- Errors after enabling Secure Sockets Layer (SSL) or SSL-related error messages

Errors when trying to configure or enable security

What kind of error are you seeing?

- ""LTPA password not set. validation failed" message displayed as error in the Administrative Console after saving global security settings "
- ""Validation failed for user userid. Please try again..." displayed in the Administrative Console after saving global security settings "
- "The setupClient.bat or setupClient.sh file is not working correctly" on page 228
- "Java HotSpot Server VM warning: Unexpected Signal 11 occurred under user-defined signal handler 0x7895710a message occurs in the native_stdout.log file when enabling security on the HP-UX11i platform" on page 228
- "WebSphere Application Server Version 6 is not working correctly with Enterprise Workload Manager (EWLM)" on page 228
- If you have successfully configured security (made changes, saved the configuration, and enabled security with no errors), but are now having problems accessing Web resources or the administrative console, refer to Errors or access problems after enabling security.

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

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

"LTPA password not set. validation failed" message displayed as error in the Administrative Console after saving global security settings

This error can be caused if, when configuring WebSphere Application Server security, ″LTPA″ is selected as the authentication mechanism, and the LTPA password field is not set. To resolve this problem:

- Select Security **Authentication Mechanism > LTPA** in the console left-hand navigation pane.
- Complete the password and confirm password fields.
- Click **OK**.
- Try setting Global Security again.

"Validation failed for user userid. Please try again..." displayed in the Administrative Console after saving global security settings

This typically indicates that a setting in the User Registry configuration is not valid:

- If the user registry is LocalOS, it is likely that either the server user ID and password are invalid or the server user ID does not have ″Act As Part of the Operating System″ (for NT) or root authority (for UNIX). The server user ID needs this authority for authentication using the LocalOS user registry.
- If the user registry is Lightweight Directory Access Protocol (LDAP):
  - Any of the settings that enable WebSphere Application Server to communicate with LDAP might be invalid, such as the LDAP server's user ID, password, host, port, or LDAP filter. When you select **Apply** or **OK** on the Global Security panel, a validation routine connects to the registry just as it would during runtime when security is enabled. This is done in order to verify any configuration problems immediately, instead of waiting until the server restarts.
  - Verify whether your LDAP server requires the Bind Distinguished Name (DN) to find the user in the LDAP directory. If the bind distinguished name is required, you must specify a DN instead of a short name. You can specify the bind distinguished name by clicking **Security > User Registries > LDAP** in the administrative console. For example, you might add `cn=root`.
  - Sometimes the LDAP server might be down during configuration. The best way to check this is to issue a command line search using a utility such as ldapsearch to search for the server ID. This way you can determine if the server is running and if the server ID is a valid entry in the LDAP. The ldapsearch utility is installed during an LDAP or Lotus Notes installation.

- If the user registry is Custom, double check that your implementation is in the classpath. Also, check to see if your implementation is authenticating properly.
- Regardless of registry type, check the User Registries configuration panels to see if you can find a configuration error:
  – Go back to the User Registries configuration panels and retype the password for the server ID.
- See if there is an obvious configuration error. Double check the attributes specified.

The setupClient.bat or setupClient.sh file is not working correctly

The `setupClient.bat` file on Windows platforms and the `setupClient.sh` file on UNIX platforms incorrectly specify the location of the SOAP security properties file.

In the `setupClient.bat` file, the correct location should be:

```
set CLIENTSOAP=-Dcom.ibm.SOAP.ConfigURL=file:%WAS_HOME%/properties/soap.client.props
```

In the `setupClient.sh` file, the CLIENTSOAP variable should be:

```
CLIENTSOAP=-Dcom.ibm.SOAP.ConfigURL=file:$WAS_HOME/properties/soap.client.props
```

In the `setupClient.bat` and `setupClient.sh` files, complete the following steps:
1. Remove the leading / after file:.
2. Change `sas` to `soap`.

Java HotSpot Server VM warning: Unexpected Signal 11 occurred under user-defined signal handler 0x7895710a message occurs in the `native_stdout.log` file when enabling security on the HP-UX11i platform

After you enable security on HP-UX 11i platforms, the following error in the `native_stdout.log` file occurs, along with a core dump and WebSphere Application Server does not start:

```
Java HotSpot(TM) Server VM warning:
Unexpected Signal 11 occurred under user-defined signal handler 0x7895710a
```

To work around this error, apply the fixes recommended by HP for Java at the following URL: http://www.hp.com/products1/unix/java/infolibrary/patches.html.

WebSphere Application Server Version 6 is not working correctly with Enterprise Workload Manager (EWLM)

To use WebSphere Application Server Version 6 with Enterprise Workload Manager (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.

Refer to Configuring server.policy files for more information on configuring server.policy files.

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.

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 signon, I cannot log on to the administrative console

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see "Obtaining help from IBM" on page 155.

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 whileinvoking 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. Please ensure that user and password are correct on the command line or in a properties file.`

  To grant an ID administrative authority, from the administrative console, click **System Administration > Console Users** and validate that the ID is a member. If it is not, add the ID with at least monitor access privileges, for read-only access.
- Check that the *enable_trusted_application* flag is set to true. To check the *enable_trusted_application* flag, from the Administrative Console, click **Security > Global Security > Custom Properties > Enable Trusted Application** and verify that it is set to true.

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 displays 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 Socket 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 which contain a statement such as: `ERROR: Could not get the initial context or unable to look up the starting context.Exiting.` followed by `javax.net.ssl.SSLHandshakeException`

The client cannot access an enterprise bean after enabling security

If client access to an enterprise bean fails after security is enabled:
- Review the steps for securing and granting access to resources.
- Browse the server 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 JavaSever Page (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 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 read by the `sas.client.props` properties file used by the client program does not have the `securityEnabled` flag set to **true**.

    To resolve this problem, make sure that the `sas.client.props` file on the client side has its securityEnabled flag set to **true**.

  Errors similar to **Authorization failed for *valid_user* while invoking *resource* securityName:/username;accessId:xxxxxx not granted any of the required roles *roles*** indicate that a client attempted to access a secured enterprise bean resource, and the supplied user ID is not assigned the required roles for that enterprise bean.
  - Check 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 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 invalid:

```
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 GUI) 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 GUI. To change the configuration for the target server, modify the inbound configuration in the Security GUI.

Similarly, an exception like org.omg.CORBA.INITIALIZE: JSAS0477W: SECURITY CLIENT/SERVER CONFIG MISMATCH: appearing on the server trying to service a client request indicates a security configuration mismatch between client and server. The steps for resolving the problem are the same as for the JSAS1477W exceptions previously described.

Client program never gets prompted when accessing secured enterprise bean

Even though it appears security is enabled and an enterprise bean is secured, it can happen that the client executes the remote method without prompting. If the remote method is protected, an authorization failure results. Otherwise, execute 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 global security settings from within the Security section of the administrative console.
- The client does not have security enabled in the sas.client.props file. Edit the sas.client.props 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 has an invalid URL syntax, or the `sas.client.props` that is pointed to cannot be found. Verify that the **com.ibm.CORBA.ConfigURL** property is valid, for example, similar to the C:/WebSphere/AppServer/properties/sas.client.props file on Windows systems. Check the Java documentation for a description of URL formatting rules. Also, validate that the file exists at the specified path.
- The client configuration does not support message layer client authentication (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 (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.sh -help command to display the parameters to use.

Use the following command options after enabling security:
- `./stopServer.sh hostname -username` *`name`* `-password` *`password`*
- `./stopNode.sh -username` *`name`* `-password` *`password`*
- `./stopManager.sh -username` *`name`* `-password` *`password`*

After enabling single signon, I cannot log on to the administrative console

This problem occurs when single signon (SSO) is enabled, and you attempt to access the administrative console using the short name of the server, for example `http://myserver:9060/ibm/console`. The server accepts your user ID and password, but returns you to the log on 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`.

Errors after enabling security

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.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see “Obtaining help from IBM” on page 155.

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 they are correct.
- **Security configuration error : User registry type is not set correctly.** Check the user registry property in global security settings in the administrative console. Verify that it 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 should have access to a resource does not, there is probably a missing configuration step. Review the steps for securing and granting access to resources.

Specifically:

- Check 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 problem user is required to have one or more of the required roles, use the administrative console to assign that user to required roles. Then 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 an exception, `SecurityException`.

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 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 run-time flag **java.security.debug**
  - 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 and what they mean , 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 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 JMS messaging 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 which 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 LTPA as the authentication mechanism, but not generating the LTPA keys. The LTPA keys encrypt the LTPA token.

To resolve this problem:

1. Click **System Administration > Console users > LTPA**
2. Enter a password, which can be anything.
3. Enter the same password in **Confirm Password**.
4. Click **Apply**.
5. Click **Generate Keys**.
6. Click on **Save**.

The exception AccessControlException, 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 Version 5. An exception similar to the following example displays. The error message and number can vary.

```
E CWSRV0020E: [Servlet Error]-[validator]:
Failed to load servlet:
java.security.AccessControlException: access denied
(java.io.FilePermission
C:\WebSphere\AppServer\installedApps\maeda\adminconsole.ear\adminconsole.war\
WEB-INF\validation.xml read)
```

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 information center navigation. 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 thrown when a client tries to access a resource 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 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 an ra.xml file bundled in a .rar file
- The class path is not set correctly, preventing the permissions for the resource.xml file for 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 in the SystemOut.log file which reports the details of the syntax error. For example:

  `CWSCJ0189E: Caught ParserException` while creating template for application policy `C:\WAS\config\cells\xxx\nodes\xxx\app.policy .`

  The exception is com.ibm.ws.security.util.ParserException: line 18: expected ';', found 'grant'
- Look for a message similar to: `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:
C:\WebSphere\AppServer\logs\server1\SystemOut_02.08.20_11.19.53.log :
access denied (java.io.FilePermission
C:\WebSphere\AppServer\logs\server1\SystemOut_02.08.20_11.19.53.log delete)

Code:
 com.ibm.ejs.ras.RasTestHelper$7  in
{file:/C:/WebSphere/AppServer/installedApps/maeda/JrasFVTApp.ear/RasLib.jar
}

Stack Trace:

java.security.AccessControlException: access denied
(java.io.FilePermission C:\WebSphere\AppServer\logs\server1
\SystemOut_02.08.20_11.19.53.log delete)
        at java.security.AccessControlContext.checkPermission
                           (AccessControlContext.java(Compiled Code))
        at java.security.AccessController.checkPermission
                           (AccessController.java(Compiled Code))
        at java.lang.SecurityManager.checkPermission
                           (SecurityManager.java(Compiled Code))
                            .
Code Base Location:

com.ibm.ws.security.core.SecurityManager :
file:/C:/WebSphere/AppServer/lib/securityimpl.jar

  ClassLoader: com.ibm.ws.bootstrap.ExtClassLoader
  Permissions granted to CodeSource
(file:/C:/WebSphere/AppServer/lib/securityimpl.jar <no certificates>

  {
    (java.util.PropertyPermission java.vendor read);
    (java.util.PropertyPermission java.specification.version read);
    (java.util.PropertyPermission line.separator read);
    (java.util.PropertyPermission java.class.version read);
    (java.util.PropertyPermission java.specification.name read);
    (java.util.PropertyPermission java.vendor.url read);
    (java.util.PropertyPermission java.vm.version read);
    (java.util.PropertyPermission os.name read);
    (java.util.PropertyPermission os.arch read);
   }
   ( This list continues.)
```

- If the method is SPI, check the resources.xml file to ensure that the 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 ra.xml file, correct them with the policytool. Avoid editing the policy manually, because syntax errors can result.
- If a permission is listed as Unresolved, it does not take effect. Verify that the specified permission name is correct.
- If the class path 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 ra.xml file.
- If the permission should not be 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 they were 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 Java 2 Security in the information center navigation carefully to determine in which policy file or ra.xml file that permission should be specified.

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 indicated is not found in the 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.

Errors trying to enable or configure Secure Socket Layer (SLL) encrypted access

What kind of error are you seeing?
- "The Java Cryptographic Extension (JCE) files were not found." error when launching iKeyman.

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

"The Java Cryptographic Extension (JCE) files were not found." error when launching iKeyman.

You may receive the following error when you attempt to start the iKeyman tool: "The Java Cryptographic Extension (JCE) files were not found. Please check that the JCE files have been installed in the correct directory". When you click OK, the iKeyman tool closes. To resolve this problem:
- Set the JAVA_HOME parameter so that is points to the Java Developer Kit that is shipped with WebSphere Application Server.
  - For example, on a Unix platform the command would be similar to: `export JAVA_HOME=/opt/WebSphere/AppServer/java`
  - On a Windows platform, if WebSphere Application Server is installed on your c: drive, the command would be: `set JAVA_HOME=c:\WebSphere\AppServer\java`
- Rename the file install_dir/java/jre/lib/ext/gskikm.jar to gskikm.jar.org.

Errors after configuring or enabling Secure Sockets Layer

This article explains various problems you might encounter after configuring or enabling Secure Sockets Layer (SSL).

Stopping the deployment manager after configuring Secure Sockets Layer

After configuring the Secure Sockets Layer repertoires, if you stop the deployment manager without also stopping the node agents, you might receive the following error message when you restart the deployment manager:

```
CWWMU0509I: The server "nodeagent" cannot be reached. It appears to be stopped.
CWWMU0211I: Error details may be seen in the file:
           /opt/WebSphere/AppServer/logs/nodeagent/stopServer.log
```

The error occurs because the deployment manager did not propagate the new SSL certificate to the node agents. Thus, the node agents are using an older certificate files than the deployment manager and the certificate files are incompatible. To work around this problem, you must manually stop the node agent and deployment manager processes. To end the processes on Windows platforms, use the Task Manager. On UNIX platforms, run the command to end the process.

There are some things you need to consider when identifying the specific process that should be killed. For each process being killed, WebSphere Application Server stores the process ID in a `pid` file and you need to find these `*.pid` files. For example, the `server1.pid` for a standalone install might be found at: `<WAS_root>/AppServer/logs/server1.pid`

Accessing resources using HTTPS

If you are unable to access resources using a Secure Sockets Layer (SSL) URL (beginning with https:), or encounter error messages which indicate SSL problems, verify that your HTTP server is configured correctly for SSL by browsing the welcome page of the HTTP server using SSL by entering the URL: **https://***hostname*.

If the page works with HTTP, but not HTTPS, the problem is with the HTTP server.

- Refer to the documentation for your HTTP server for instructions on correctly enabling SSL. If you are using the IBM HTTP Server or Apache, go to:http://www.ibm.com/software/webservers/httpservers/library.html. Click **Frequently Asked Questions**> **SSL**.
- If you are use the IBM Key Management (IKeyman) tool to create certificates and keys, remember to stash the password to a file when creating the KDB file with the IBM Key Management Tool.
  1. Go to the directory where the KDB file was created, and see if there is a .sth file.
  2. If not, open the KDB file with the IBM Key Management Tool, and click **Key Database File > Stash Password**. The following message displays: `The password has been encrypted and saved in the file`.

If the HTTP server handles SSL-encrypted requests successfully, or is not involved (for example, traffic flows from a Java client application directly to an enterprise bean hosted by the WebSphere Application Server, or the problem appears only after enabling WebSphere Application Server security), what kind of error are you seeing?

- `javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: handshake failure`
- `javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: unknown certificate`
- `javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: bad certificate`
- `org.omg.CORBA.INTERNAL: EntryNotFoundException or NTRegistryImp E CWSCJ0070E: No privilege id configured for:` error when programmatically creating a credential.

For general tips on diagnosing and resolving security-related problems, see “Security components troubleshooting tips” on page 240

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see “Obtaining help from IBM” on page 155

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. Click **WebSphere Administrative Console Security Settings > SSL Configuration Repertoires > DefaultSSLSettings** (or other named SSL settings).
2. Select the **Secure Sockets Layer (SSL)** option from the Additional Properties menu. You can also browse the file manually by viewing the *install_dir*/properties/sas.client.props file.
3. Check the property specified by the com.ibm.ssl.protocol file to determine which protocol is specified.
4. Check the cipher types specified by the com.ibm.ssl.enabledCipherSuites. You might want to add more cipher types to the list. To see which cipher suites are currently enabled: Fo to the properties page of the SSL settings as described above, and look for the **Cipher Suites** property. To see the list of all possible cipher suites, go to the properties page of the SSL settings as described above, then view the online help for that page. From the help page, click **Configure additional SSL settings**.
5. Correct the protocol or cipher problem by using a different client or server protocol and cipher selection. Typical protocols are SSL or SSLv3.
6. Make the cipher selection 40-bit instead of 128-bit. For 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 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

If you see a Java exception stack similar to the following example, it can be caused by having a personal certificate in the client keystore used for SSL mutual authentication but not having extracted the signer certificate into the server truststore file so that the server can trust it whenever the SSL handshake is made:

```
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.

Security components troubleshooting tips

This document explains basic resources and steps for diagnosing security related issues in the WebSphere Application Server, including:

- What “Log files” to look at and what to look for in them.
- What “SDSF output logs” on page 242 to look at and what to look for in them.
- “General approach for troubleshooting security-related issues” on page 243 to isolating and resolving security problems.
- When and how to “Trace security” on page 247.
- An overview and table of “CSIv2 CORBA Minor Codes” on page 249.

The following security-related problems are addressed elsewhere in the information center:
- Errors and access problems after enabling security

  After enabling global security, there was a degradation in performance. See Enabling global security for information about using the unrestricted policy files.
- Errors after enabling SSL, or SSL-related error messages
- Errors trying to configure and enable security

If none of these steps solves the problem, check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning.

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

Note: For an overview of WebSphere Application Server security components such as SAS or z/SAS and how they work, see Getting started with security.

Log files

When troubleshooting the security component, browse the 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 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 SSL configuration might lead to the following message. You should ensure that the keystore location and keystore passwords are valid. Also, ensure the keystore has a valid personal certificate and that the personal certificate public key or CA root has been extracted on put into the truststore.

```
SASRas        A 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
```

SDSF output logs

When troubleshooting the security component, browse the SDSF 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:

```
BBOM0001I com_ibm_Server_Security_Enabled: 1.
BBOM0001I com_ibm_CSI_claimTLClientAuthenticationSupported: 1.
BBOM0001I com_ibm_CSI_claimTLClientAuthenticationRequired: 0.
BBOM0001I com_ibm_CSI_claimTransportAssocSSLTLSSupported: 1.
BBOM0001I com_ibm_CSI_claimTransportAssocSSLTLSRequired: 0.
BBOM0001I com_ibm_CSI_claimMessageConfidentialityRequired: 0.
BBOM0001I com_ibm_CSI_claimClientAuthenticationSupported: 1.
BBOM0001I com_ibm_CSI_claimClientAuthenticationRequired: 0.
BBOM0001I com_ibm_CSI_claimClientAuthenticationtype:
SAFUSERIDPASSWORD.
```

```
BBOM0001I com_ibm_CSI_claimIdentityAssertionTypeSAF: 0.
BBOM0001I com_ibm_CSI_claimIdentityAssertionTypeDN: 0.
BBOM0001I com_ibm_CSI_claimIdentityAssertionTypeCert: 0.
BBOM0001I com_ibm_CSI_claimMessageIntegritySupported: NOT SET,DEFAULT=1.
BBOM0001I com_ibm_CSI_claimMessageIntegrityRequired: NOT SET,DEFAULT=1.
BBOM0001I com_ibm_CSI_claimStateful: 1.
BBOM0001I com_ibm_CSI_claimSecurityLevel: HIGH.
BBOM0001I com_ibm_CSI_claimSecurityCipherSuiteList: NOT SET.
BBOM0001I com_ibm_CSI_claimKeyringName: WASKeyring.
BBOM0001I com_ibm_CSI_claim_ssl_sys_v2_timeout: NOT SET, DEFAULT=100.
BBOM0001I com_ibm_CSI_claim_ssl_sys_v3_timeout: 600.
BBOM0001I com_ibm_CSI_performTransportAssocSSLTLSSupported: 1.
BBOM0001I security_sslClientCerts_allowed: 0.
BBOM0001I security_kerberos_allowed: 0.
BBOM0001I security_userid_password_allowed: 0.
BBOM0001I security_userid_passticket_allowed: 1.
BBOM0001I security_assertedID_IBM_accepted: 0.
BBOM0001I security_assertedID_IBM_sent: 0.
BBOM0001I nonauthenticated_clients_allowed: 1.
BBOM0001I security_remote_identity: WSGUEST.
BBOM0001I security_local_identity: WSGUEST.
BBOM0001I security_EnableRunAsIdentity: 0.

 Messages beginning with BBOO0222I are common to Java WebSphere security.
They appear in both the controller and servant but are applicable to the servant.

 BBOO0222I CWSCJ0240I: Security service initialization completed successfully
 BBOO0222I CWSCJ0215I: Successfully set JAAS login provider
 configuration class to com.ibm.ws.security.auth.login.Configuration.
 BBOO0222I CWSCJ0136I: Custom
 Registry:com.ibm.ws.security.registry.zOS.SAFRegistryImpl has been  initialized
 BBOO0222I CWSCJ0157I: Loaded Vendor AuthorizationTable:
 com.ibm.ws.security.core.SAFAuthorizationTableImpl
 BBOO0222I CWSCJ0243I: Security service started successfully
 BBOO0222I CWSCJ0210I: Security enabled true
```

General approach for troubleshooting security-related issues

When troubleshooting security-related problems, the following questions are very helpful and should be considered:

Does the problem occur when security is disabled?

This is a good litmus test to determine that a problem is security related. However, just because a problem only occurs when security is enabled does not always make it a security problem. More troubleshooting is necessary to ensure the problem is really security-related.

Did security appear to initialize properly?

A lot of security code is visited during initialization. So you will likely see problems there first if the problem is configuration related.

The following sequence of messages generated in the SystemOut.log indicate normal code initialization of an application server. This sequence will vary 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: 2003/08/25 13:06:31.034 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.auth.login.Configuration
   SourceId: com.ibm.ws.security.auth.login.Configuration
   Category: AUDIT
   ExtendedMessage: CWSCJ0215I: Successfully set JAAS login provider
   configuration class to com.ibm.ws.security.auth.login.Configuration.
 Trace: 2003/08/25 13:06:31.085 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.SecurityDM
   SourceId: com.ibm.ws.security.core.SecurityDM
   Category: INFO
   ExtendedMessage: CWSCJ0231I: The Security component's
   FFDC Diagnostic Module com.ibm.ws.security.core.SecurityDM
   registered success
 fully: true.
 Trace: 2003/08/25 13:06:31.086 01 t=9EA930 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 812
   error message: BBOO0222I CWSCJ0231I: The Security component's
FFDC Diagnostic Module com.ibm.ws.security.core.SecurityDM registered
  successfully: true.
 Trace: 2003/08/25 13:06:32.426 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.SecurityComponentImpl
   SourceId: com.ibm.ws.security.core.SecurityComponentImpl
   Category: INFO
   ExtendedMessage: CWSCJ0309I: Java 2 Security is disabled.
 Trace: 2003/08/25 13:06:32.427 01 t=9EA930 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 812
   error message: BBOO0222I CWSCJ0309I: Java 2 Security is disabled.
 Trace: 2003/08/25 13:06:32.445 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.SecurityComponentImpl
   SourceId: com.ibm.ws.security.core.SecurityComponentImpl
   Category: INFO
   ExtendedMessage: CWSCJ0212I: WCCM JAAS configuration information
successfully pushed to login provider class.
 Trace: 2003/08/25 13:06:32.445 01 t=9EA930 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 812
   error message: BBOO0222I CWSCJ0212I: WCCM JAAS configuration
 information successfully pushed to login provider class.
 Trace: 2003/08/25 13:06:32.459 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: SecurityComponentImpl
   SourceId: SecurityComponentImpl
   Category: WARNING
   ExtendedMessage: BBOS1000W  LTPA or ICSF are configured as the
authentication mechanism but SSO is disabled.
 Trace: 2003/08/25 13:06:32.459 01 t=9EA930 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 824
   error message: BBOS1000W  LTPA or ICSF are configured as the
```

```
authentication mechanism but SSO is disabled.
 Trace: 2003/08/25 13:06:32.463 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.SecurityComponentImpl
   SourceId: com.ibm.ws.security.core.SecurityComponentImpl
   Category: INFO
   ExtendedMessage: CWSCJ0240I: Security service initialization completed
successfully
 Trace: 2003/08/25 13:06:32.463 01 t=9EA930 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 812
   error message: BBOO0222I CWSCJ0240I: Security service initialization
completed successfully
 Trace: 2003/08/25 13:06:39.718 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.registry.UserRegistryImpl
   SourceId: com.ibm.ws.security.registry.UserRegistryImpl
   Category: AUDIT
   ExtendedMessage: CWSCJ0136I: Custom Registry:
com.ibm.ws.security.registry.zOS.SAFRegistryImpl has been initialized
 Trace: 2003/08/25 13:06:41.967 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.WSAccessManager
   SourceId: com.ibm.ws.security.core.WSAccessManager
   Category: AUDIT
   ExtendedMessage: CWSCJ0157I: Loaded Vendor AuthorizationTable:
com.ibm.ws.security.core.SAFAuthorizationTableImpl
 Trace: 2003/08/25 13:06:43.136 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.role.RoleBasedAuthorizerImpl
   SourceId: com.ibm.ws.security.role.RoleBasedAuthorizerImpl
   Category: AUDIT
   ExtendedMessage: CWSCJ0157I: Loaded Vendor AuthorizationTable:
com.ibm.ws.security.core.SAFAuthorizationTableImpl
 Trace: 2003/08/25 13:06:43.789 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.SecurityComponentImpl
   SourceId: com.ibm.ws.security.core.SecurityComponentImpl
   Category: INFO
   ExtendedMessage: CWSCJ0243I: Security service started successfully
 Trace: 2003/08/25 13:06:43.789 01 t=9EA930 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 812
   error message: BBOO0222I CWSCJ0243I: Security service started successfully
 Trace: 2003/08/25 13:06:43.794 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.SecurityComponentImpl
   SourceId: com.ibm.ws.security.core.SecurityComponentImpl
   Category: INFO
   ExtendedMessage: CWSCJ0210I: Security enabled true
 Trace: 2003/08/25 13:06:43.794 01 t=9EA930 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 812
   error message: BBOO0222I CWSCJ0210I: Security enabled true
 Trace: 2003/08/25 13:07:06.474 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.WSAccessManager
   SourceId: com.ibm.ws.security.core.WSAccessManager
   Category: AUDIT
   ExtendedMessage: CWSCJ0157I: Loaded Vendor AuthorizationTable:
com.ibm.ws.security.core.SAFAuthorizationTableImpl
 Trace: 2003/08/25 13:07:09.315 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.WSAccessManager
   SourceId: com.ibm.ws.security.core.WSAccessManager
   Category: AUDIT
   ExtendedMessage: CWSCJ0157I: Loaded Vendor AuthorizationTable:
com.ibm.ws.security.core.SAFAuthorizationTableImpl
```

Is there a stack trace or exception printed in the SystemOut.log?

A single stack trace tells a lot about the problem. What code initiated the code that failed? What is the failing component? Which class did the failure actually come from? Sometimes the stack trace

is all that is needed to solve the problem and it can pinpoint the root cause. Other times, it can only give us a clue, and could actually be misleading. When support analyzes a stack trace, they may request additional trace if it is not clear what the problem is. If it appears to be security-related and the solution cannot be determined from the stack trace or problem description, you will be asked to gather the following trace specification:
`SASRas=all=enabled:com.ibm.ws.security.*=all=enabled` from all processes involved.

Is this a distributed security problem or a local security problem?

- If the problem is local, that is the code involved does not make a remote method invocation, then troubleshooting is isolated to a single process. It is important to know when a problem is local versus distributed since the behavior of the ORB, among other components, is different between the two. Once a remote method invocation takes place, an entirely different security code path is entered.
- When you know that the problem involves two or more servers, the techniques of troubleshooting change. You will need to trace all servers involved simultaneously so that the trace shows the client and server sides of the problem. Try to make sure the timestamps on all machines match as closely as possible so that you can find the request and reply pair from two different processes. Enable both SAS 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 is related to either the authentication protocol, authentication mechanism or user registry. When authorization fails, this is usually related to the application bindings from assembly and/or deployment and to the caller's identity who is accessing the method and the roles required by the method.

Is this a Web or EJB request?

Web requests have a completely different code path than EJB requests. Also, there are different security features for Web requests than for EJB requests, requiring a completely different body of knowledge to resolve. For example, when using the LTPA authentication mechanism, the single signon feature (SSO) is available for Web requests but not for EJB requests. Web requests involve HTTP header information not required by EJB requests due to the protocol differences. Also, the Web container (or servlet engine) is involved in the entire process. Any of these components could be involved in the problem and all should be considered during troubleshooting, based on the type of request and where the failure occurs.

Secure EJB requests heavily involve the ORB and Naming components since they flow over the RMI/IIOP protocol. In addition, when work flow management (WLM) is enabled, other behavior changes in the code can be observed. All of these components interact closely for security to work properly in this environment. At times, trace in any or all of these components might be necessary to troubleshoot problems in this area. The trace specification to begin with is `SASRas=all=enabled:com.ibm.ws.security.*=all=enabled`. ORB trace is also very beneficial when the SAS/Security trace does not seem to pinpoint the problem.

Does the problem seem to be related to the Secure Sockets Layer (SSL)?

The Secure Socket Layer (SSL) is a totally distinct separate layer of security. Troubleshooting SSL problems are usually separate from troubleshooting authentication and/or authorization problems. There are many things to consider. Usually, SSL problems are first time setup problems because the configuration can be difficult. Each client must contain the server's signer certificate. During mutual authentication, each server must contain the client's signer certificate. Also, there can be protocol differences (SSLv3 vs. TLS), and listener port problems related to stale IORs (i.e., IORs from a server reflecting the port prior to the server restarting).

For SSL problems, we sometimes request an SSL trace to determine what is happening with the SSL handshake. The SSL handshake is the process which occurs when a client opens a socket to a server. If anything goes wrong with the key exchange, cipher exchange, etc. the handshake will fail and thus the socket is invalid. Tracing JSSE (the SSL implementation used in WebSphere Application Server) involves the following steps:

- Set the following system property on the client and server processes: `-Djavax.net.debug=true`. For the server, add this to the Generic JVM Arguments property of the Java virtual machine settings page.
- Turn on ORB trace as well.
- Recreate the problem.

  The `SystemOut.log` of both processes should contain the JSSE trace. You will find trace similar to the following:

```
SSLConnection: install <com.ibm.sslite.e@3ae78375>
>> handleHandshakeV2 <com.ibm.sslite.e@3ae78375>
>> handshakeV2 type = 1
>> clientHello: SSLv2.
SSL client version: 3.0
...
...
...
JSSEContext: handleSession[Socket[addr=null,port=0,localport=0]]

<< sendServerHello.
SSL version: 3.0
SSL_RSA_WITH_RC4_128_MD5
HelloRandom
...
...
...
<< sendCertificate.
<< sendServerHelloDone.
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleHandshake <com.ibm.sslite.e@3ae78375>
>> handshakeV3 type = 16

>> clientKeyExchange.
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleChangeCipherSpec <com.ibm.sslite.e@3ae78375>
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleHandshake <com.ibm.sslite.e@3ae78375>
>> handshakeV3 type = 20
>> finished.
<< sendChangeCipherSpec.
<< sendFinished.
```

Trace security

The classes which implement WebSphere Application Server security are:

- com.ibm.ws.security.*
- com.ibm.websphere.security.*
- com.ibm.WebSphereSecurityImpl.*
- SASRas

To view detailed information on the 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 GUI.
- `SASRas=all=enabled`. This trace statement collects the trace for SAS (low-level authentication logic).

Fine tuning SAS traces:

If a subset of classes need to be traced for the SAS/CSIv2 component, a system property can be specified with the class names comma separated:
`com.ibm.CORBA.securityTraceFilter=SecurityConnectionInterceptorImpl, VaultImpl, ...`

Fine tuning Security traces:

If a subset of packages need to be traced, specify a trace specification more detailed than `com.ibm.ws.security.*=all=enabled`. For example, to trace just dynamic policy code, you can specify `com.ibm.ws.security.policy.*=all=enabled`. To disable dynamic policy trace, you can specify `com.ibm.ws.security.policy.*=all=disabled`.

Configuring CSIv2, or z/SAS Trace Settings

Situations arise where reviewing trace for the CSIv2 and z/SAS authentication protocols can assist in troubleshooting difficult problems. This section describes how to enable to CSIv2 and z/SAS trace.

Enabling Server-Side CSIv2 and z/SAS Trace

To enable z/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`.

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.
- In this file, change `traceFileName=` to point to the path in which you want the ouput file created. Make sure you put a double backslash (\\) between each subdirectory. For example, `traceFileName=c:\\WebSphere\\AppServer\\logs\\sas_client.log`
- In this file, add the trace specification string: `SASRas=all=enabled`. Any additional trace strings can be added on separate lines.
- Point to this file from within your client application. On the Java command line where you launch the client, add the following system property:
  `-DtraceSettingsFile=TraceSettings.properties`.

  Note: Do not give the fully qualified path to the `TraceSettings.properties` file. Make sure that the `TraceSettings.properties` file is in your 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

Whatever exceptions might occur within the security code on either the client or server, the eventual exception will become a CORBA exception. So any exception that occurs gets ″wrapped″ by a CORBA exception, because the CORBA architecture is used by the security service for its own inter-process communication. CORBA exceptions are generic, and indicate a problem in communication between two components. CORBA minor codes are more specific, and indicate the underlying reason that a component could not complete a request.

The following shows the CORBA Minor codes which a client can expect to receive after executing a security-related request such as authentication. It also includes the CORBA exception type that the minor code would appear in.

The following exception shows an example of a CORBA exception where the minor code is 49424300. From the table below, this minor code indicates Authentication Failure. Typically, a descriptive message is also included in the exception to assist in troubleshooting the problem. Here, the detailed message is ″Exception caught invoking authenticateBasicAuthData from SecurityServer for user jdoe. Reason: com.ibm.WebSphereSecurity.AuthenticationFailedException″ which indicates that the authentication failed for user ″jdoe″.

The completed field in the exception indicates whether the method was completed or not. In the case of a NO_PERMISSION, the method should never get invoked, so it will always be ″completed:No″. Other exceptions which are caught on the server side could have a completed status of ″Maybe″ or ″Yes″.

```
org.omg.CORBA.NO_PERMISSION: Caught WSSecurityContextException in
WSSecurityContext.acceptSecContext(),
reason: Major Code[0] Minor Code[0] Message[Exception caught invoking
authenticateBasicAuthData from SecurityServer for user jdoe.  Reason:
com.ibm.WebSphereSecurity.AuthenticationFailedException]  minor code: 49424300
completed: No

at com.ibm.ISecurityLocalObjectBaseL13Impl.PrincipalAuthFailReason.
map_auth_fail_to_minor_code(PrincipalAuthFailReason.java:83)
        at com.ibm.ISecurityLocalObjectBaseL13Impl.CSIServerRI.receive_request
               (CSIServerRI.java:1569)
        at com.ibm.rmi.pi.InterceptorManager.iterateReceiveRequest
              (InterceptorManager.java:739)
        at com.ibm.CORBA.iiop.ServerDelegate.dispatch(ServerDelegate.java:398)
        at com.ibm.rmi.iiop.ORB.process(ORB.java:313)
        at com.ibm.CORBA.iiop.ORB.process(ORB.java:1581)
        at com.ibm.rmi.iiop.GIOPConnection.doWork(GIOPConnection.java:1827)
        at com.ibm.rmi.iiop.WorkUnitImpl.doWork(WorkUnitImpl.java:81)
        at com.ibm.ejs.oa.pool.PooledThread.run(ThreadPool.java:91)
        at com.ibm.ws.util.CachedThread.run(ThreadPool.java:149)
```

The following table shows the CORBA Minor codes which a client can expect to receive after executing a security-related request such as authentication. It also includes the CORBA exception type that the minor code would appear in.

Minor code name	Minor code value (in hex)	Exception type (all in the package of org.omg.CORBA .*)	Minor code description	Retry performed (when authenticationRe tryEnabled=true)
AuthenticationFailed	49424300	NO_PERMISSION	This is a generic authentication failed error. It does not give any details about whether the userid or password is invalid. Some registries can choose to use this type of error code, others might choose to use the next three types which are more specific.	Yes

InvalidUserid	49424301	NO_PERMISSION	This occurs when the registry returns bad userid.	Yes
InvalidPassword	49424302	NO_PERMISSION	This occurs when the registry returns bad password.	Yes
InvalidSecurityCredentials	49424303	NO_PERMISSION	This is a generic error indicating that the credentials are bad for whatever reason. It could be that they don't have the right attributes set.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).
InvalidRealm	49424304	NO_PERMISSION	This occurs when the REALM in the token received from the client does not match the server's current realm.	No
ValidationFailed	49424305	NO_PERMISSION	A validation failure occurs when a token is sent from the client or server to a target server but the token format or the expiration is invalid.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).
CredentialTokenExpired	49424306	NO_PERMISSION	This is more specific about why the validation failed. In this case, the token has a absolute lifetime, and this lifetime has expired. Therefore, it is no longer a valid token and cannot be used.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).
InvalidCredentialToken	49424307	NO_PERMISSION	This is more specific about why the validation failed. In this case, the token cannot be decrypted or the data within it is not readable.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).
SessionDoesNotExist	49424308	NO_PERMISSION	This indicates that the CSIv2 session does not exist on the server. Typically, a retry occurs automatically and will successfully create a new session.	Yes
SessionConflictingEvidence	49424309	NO_PERMISSION	This indicates that a session already exists on the server which matches the context_id sent over by the client, however, the information provided by the client for this EstablishContext message is different from the information originally provided to establish the session.	Yes
SessionRejected	4942430A	NO_PERMISSION	This indicates that the session referenced by the client has been previously rejected by the server.	Yes
SecurityServerNotAvailable	4942430B	NO_PERMISSION	This error occurs when the server cannot contact the security server (whether local or remote) in order to authenticate or validate.	No
InvalidIdentityToken	4942430C	NO_PERMISSION	This error indicates that identity cannot be obtained from the identity token when Identity Assertion is enabled.	No

IdentityServerNotTrusted	4942430D	NO_PERMISSION	This indicates that the server id of the sending server is not on the target server's trusted principal list.	No
InvalidMessage	4942430E	NO_PERMISSION	This indicates that the CSIv2 message format is invalid for the receiving server.	No
AuthenticationNotSupported	49421090	NO_PERMISSION	This error occurs when a mechanism does not support authentication (very rare).	No
InvalidSecurityMechanism	49421091	NO_PERMISSION	This is used to indicate that the specified security mechanism is not known.	No
CredentialNotAvailable	49421092	NO_PERMISSION	This indicates a credential is not available when it is required.	No
SecurityMechanismNotSupported	49421093	NO_PERMISSION	This error occurs when a security mechanism specified in the CSIv2 token is not implemented on the server.	No
ValidationNotSupported	49421094	NO_PERMISSION	This error occurs when a mechanism does not support validation (such as LocalOS). This error should not occur since the LocalOS credential is not a forwardable credential, therefore, validation should never need to be called on it.	No
CredentialTokenNotSet	49421095	NO_PERMISSION	This is used to indicate the token inside the credential is null.	No
ServerConnectionFailed	494210A0	COMM_FAILURE	This error is used when a connection attempt fails.	Yes (via ORB retry)
CorbaSystemException	494210B0	INTERNAL	This is a generic CORBA specific exception in system code.	No
JavaException	494210B1	INTERNAL	This is a generic error that indicated an unexpected Java exception occurred.	No
ValueIsNull	494210B2	INTERNAL	This is used to indicate that a value or parameter passed in was null.	No
EffectivePolicyNotPresent	494210B3	INTERNAL	This indicates that an effective policy object for CSIv2 is not present. This object is used to determine what security configuration features have been specified.	No
NullPointerException	494210B4	INTERNAL	This is used to indicate that a NullPointerException was caught in the runtime.	No
ErrorGettingClassInstance	494210B5	INTERNAL	This indicates a problem loading a class dynamically.	No
MalFormedParameters	494210B6	INTERNAL	This indicates parameters are not valid.	No
DuplicateSecurityAttributeType	494210B7	INTERNAL	A duplicate credential attribute has been specified during the set_attributes operation.	No
MethodNotImplemented	494210C0	NO_IMPLEMENT	A method invoked has not been implemented.	No

GSSFormatError	494210C5	BAD_PARAM	This indicates that a GSS encoding or decoding routine has thrown an exception.	No
TagComponentFormatError	494210C6	BAD_PARAM	This indicates that a tag component cannot be read properly.	No
InvalidSecurityAttributeType	494210C7	BAD_PARAM	This indicates an attribute type specified during the set_attributes operation is an invalid type.	No
SecurityConfigError	494210CA	INITIALIZE	A problem exists between the client and server configuration.	No

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.

Naming and directory

Troubleshooting 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 Name space logical view and Lookup names support in deployment descriptors and thin clients.
2. Review the programming examples that are included in the sections explaining the JNDI and CosNaming interfaces.
3. Read “Naming services component troubleshooting tips” on page 39 for additional general information.
4. If you “Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP file, or other client” on page 40, read this article.

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 article 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 254 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:

cell	DumpNameSpace default. 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. The `node` value is synonymous with `host`.

For WebSphere Application Servers Version 4.0 or later:

legacy	DumpNameSpace default. Dumps the tree starting at the legacy root context.
host	Dumps the tree starting at the bootstrap host root context. The `host` value is synonymous with `node`.
tree	Dumps the tree starting at the tree root context.

For all WebSphere Application Servers 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` choice that is compatible with WebSphere Application Servers prior to Version 4.0 and 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 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
================================================================================
```

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 *install_root*/logs/trace 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 reply example 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 reply example 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 reply example, 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]
```

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 “WTRN”. 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 in the activityerror log. Log messages produced by the Transaction service are accompanied by Log Analyzer descriptions.
3. Check for more messages in 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.

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 WebSphere Application Server application programming interface reference information (Javadoc).

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 an introduction to each WebSphere extension.

ActivitySessions	How do I?...	Overview		Samples
Application profiling	How do I?...	Overview		Samples
Asynchronous beans	How do I?...	Overview		Samples
Dynamic caching	How do I?...	Overview		
Dynamic query	How do I?...	Overview		Samples
Internationalization	How do I?...	Overview		Samples
Object pools	How do I?...	Overview		
Scheduler	How do I?...	Overview		Samples

Startup beans	How do I?...	Overview		
Work areas	How do I?...	Overview		

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 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.

   Related concepts

   The ActivitySession service
   The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. An ActivitySession context can be longer-lived than a global transaction context and can encapsulate global transactions.

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

Complete the steps below to resolve problems that you think are related to the dynamic cache service.

1. For distributed platforms, review the JVM logs for your application server. Messages in either of these logs prefaced with *DYNA* result from dynamic cache service operations.
   a. On distributed platforms, view the JVM logs for your application server. Each server has its own JVM log file. For example, if your server is named *Member_1*, the JVM log is located in the

subdirectory *install_root*/logs/Member_1. To use the administration console to review the JVM logs, click **Troubleshooting > Logs and trace > *server_name* > JVM logs > Runtime > View**.

b. 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".
```

c. 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.

d. 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.

e. 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.
3. If you have completed the preceding steps and still cannot resolve the problem, contact your IBM software support representative. On distributed platforms, you can use the collector tool (collector.bat or collector.sh located in the `bin` directory) to gather trace information and other configuration information for the support team to diagnose the problem. The collector tool gathers dynamic cache service files and packages them into a JAR file. The IBM representative can specify when and where to send the JAR file. 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 page of the administrative console, select the **Enable servlet caching** check box.

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`.

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 (IHS) plug-in

Explanation The `DynaCacheEsi.ear` file is required to send invalidations to external caches.
Recommended response Install `DynaCacheEsi.ear` using the administrative console.

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.

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 Cache monitor for monitoring contents in memory cache, disk cache and external caches (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 See Displaying cache information and set the `esiInvalidationMonitor` property to `true` in the `plugin-cfg.xml` file.

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

      ```
      WebSphere/AppServer/profiles/your_profile_name/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.

Internationalization

Internationalization service errors

The internationalization service issues one exception: java.lang.IllegalStateException. This 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 "
- "The service is not started"
- "Invalid context element" on page 270
- "Missing context element" on page 270
- "Invalid policy" on page 271
- "Missing policy" on page 271

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 "Enabling tracing and logging" on page 132.

The trace strings for internationalization 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 run-time 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 run-time problems:

No ORB support

The service cannot obtain an instance of the object request broker (ORB). This condition is a fatal error. Examine the logs 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 logs 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. Review the logs for error conditions related to the activity service.

No AsynchBeans support
The service cannot register into the asynchronous beans environment. This warning indicates that the asynchronous beans environment cannot support internationalization context. If the application server is supposed to support asynchronous beans, verify that the asynchbeans.jar and asynchbeansimpl.jar files exist in the class path, and review the trace log for any error conditions related to asynchronous beans.

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 Meta-data support
The service cannot register with the meta-data service. This warning indicates that the internationalization service cannot process the internationalization policies within application deployment descriptors. Without meta-data 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 meta-data 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. For more information see Administering the internationalization service.

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 Assembling internationalized applications.

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 meta-data support is unavailable, the service logs a message and applies the policy `[CMI, RunAsCaller]` on every servlet service method and EJB business method invocation.

For more information, see the following topics:
- Assembling internationalized applications
- Container internationalization attributes
- Internationalization type
- Migrating internationalized applications

Related concepts

"Enabling tracing and logging" on page 132

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

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

Trademarks and service marks

For trademark attribution, visit the IBM Terms of Use Web site (http://www.ibm.com/legal/us/).