IBM WebSphere Application Server for z/OS, Version 8.5

Troubleshooting and support

GA32-1075-00

Note

Before using this information, be sure to read the general information under “Notices” on page 339.

Compilation date: June 7, 2012

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 email form appears.
  3. Fill out the email form as instructed, and submit your feedback.
- To send comments on PDF books, you can email your comments to: **wasdoc@us.ibm.com**.

  Your comment should pertain to specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. 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.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer. When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about your comments.

Using this PDF

Links

Because the content within this PDF is designed for an online information center deliverable, you might experience broken links. You can expect the following link behavior within this PDF:

- Links to Web addresses beginning with http:// work.
- Links that refer to specific page numbers within the same PDF book work.
- The remaining links will *not* work. You receive an error message when you click them.

Print sections directly from the information center navigation

PDF books are provided as a convenience format for easy printing, reading, and offline use. The information center is the official delivery format for IBM WebSphere Application Server documentation. If you use the PDF books primarily for convenient printing, it is now easier to print various parts of the information center as needed, quickly and directly from the information center navigation tree.

To print a section of the information center navigation:

1. Hover your cursor over an entry in the information center navigation until the **Open Quick Menu** icon is displayed beside the entry.
2. Right-click the icon to display a menu for printing or searching your selected section of the navigation tree.
3. If you select **Print this topic and subtopics** from the menu, the selected section is launched in a separate browser window as one HTML file. The HTML file includes each of the topics in the section, with a table of contents at the top.
4. Print the HTML file.

For performance reasons, the number of topics you can print at one time is limited. You are notified if your selection contains too many topics. If the current limit is too restrictive, use the feedback link to suggest a preferable limit. The feedback link is available at the end of most information center pages.

Chapter 1. How do I troubleshoot?

Follow these shortcuts to get started quickly with popular tasks.

When you visit a task in the information center, look for the **IBM® Suggests** feature at the bottom of the page. Use it to find available tutorials, demonstrations, presentations, developerWorks® articles, Redbooks®, support documents, and more.

Add tracing and logging to your applications

* For more detailed information on enabling traces by using scripting, see the Troubleshooting with scripting chapter in the *Administering applications and their environment* PDF book.

Chapter 13, “Choosing and using diagnosis tools and controls on z/OS,” on page 179

Debug WebSphere® applications during development

Detect hung threads

Detect product configuration file problems

Set traces and logs with the console

Set traces and logs with scripting*

Work with message logs

Using IBM Support Assistant

Using HPEL to troubleshoot applications

Using JSR47 for logging

Using JSR47 for logging: Configuring access logs

Chapter 2. Liberty profile: Troubleshooting tips

Tips for troubleshooting the Liberty profile.

To help you identify and resolve problems, the product has a unified logging component. See “Liberty profile: Trace and logging” on page 6.

If you receive an exception message, information about the message is available in “Liberty profile: Messages” on page 9.

The Java API document for each Liberty profile API is detailed in the Programming Interfaces (APIs) section of the information center, and is also available as a JAR file under the `/dev/ibm-api/javadoc` directory of the server image.

For details of the main known restrictions that apply when using the Liberty profile, see “Liberty profile: Runtime environment known restrictions” on page 7.

Here is a set of tips to help you troubleshoot commonly experienced problems:
- Troubleshooting JDKs
- “Troubleshooting security”
- “Troubleshooting LDAP” on page 5
- “Troubleshooting SSL” on page 5

Check that your JDKs are at a supported level

If you experience problems that are not readily explained, check that the JDKs you are using are compliant with Java Version 6 or later, and are at a current service level. See “Minimum supported Java levels” on page 7.

Troubleshooting security

This section describes some common security problems and solutions you can choose.

SESN0008E: A user authenticated as anonymous has attempted to access a session owned by user:LdapRegistry/cn=steven,o=myCompany,c=US.

This exception happens when an unauthenticated user tries to access a session created by an authenticated user. Session security which is enabled by default prevents unauthorized access of the sessions. Only the user who created a session can access it. See session security for more information.

This exception can happen when you use a JSP (`login.jsp` for example) for your form-login and the SSO token sent by the browser is expired. Because the SSO token is expired, the user is prompted to log in again using the `login.jsp` page configured for the form-login. The jsp page, by default, tries to get a session. This session was originally created by the user whose token is expired. However, since the token is expired the authentication of the user did not succeed and hence there are no credentials established when accessing this session resulting in this exception. To avoid this exception, either restart the browser which starts a new session, or configure the `login.jsp` file to not create the session by default. If you choose to update the `login.jsp` file, set `<%@ page session="false" %>`.

CWWKS9104A: Authorization failed for user {0} while invoking {1} on {2}. The user is not granted access to any of the required roles: {3}.

This error occurs when the user does not have authorization to the roles required by the application. Make sure that the user or the group it belongs to is mapped to at least one of the roles mentioned in the error message. Recall that a user-to-role mapping defined in the

ibm-application-bnd.xmi/xml file takes precedence over a mapping defined in the server.xml file. Check both resources to ensure that the correct mapping is defined. See Configuring authorization for applications on the Liberty profile.

CWWKZ0013E: It is not possible to start two applications called {0} followed by unexpected security behavior and error messages such as CWWKS9104A.

This error occurs when you have specified your application in both the server.xml (using the application element) and in the dropins folder. As a result, the application is attempted to be installed twice and the security configuration present in the server.xml file might or might not take effect. To fix this, you must remove your application from the dropins folder and copy it to another directory. If you have to leave it in the dropins folder, then you must also disable application monitoring by setting the following in your server.xml file:

```
<applicationMonitor dropinsEnabled="false"/>
```

An unauthenticated user was able to access my servlet or JSP.

A user with a principal of **UNAUTHENTICATED** (or the unauthenticated SAF user on z/OS®) is created when authentication fails or when your servlet or JSP is unprotected. An unauthenticated user is able to access your servlet or JSP if you do not define any security constraints or if you map the special subject of **EVERYONE** to the role required by your application. Review the user-to-role mappings in both the ibm-application-bnd.xmi/xml and the server.xml files. Consider the following options:

- If your servlet or JSP is unprotected, then consider adding security constraints to the deployment descriptor of your application. See Liberty profile: Authentication.
- If you do not want any unauthenticated user to access your application, then consider removing the **EVERYONE** special subject from the mapping for that role. See Configuring authorization for applications on the Liberty profile.
- If you believe that a certain user cannot be authorized to your servlet or JSP, then review who is mapped to that role by examining both the ibm-application-bnd.xmi/xml and the server.xml files. You can map a role to a user, group, or special subject. If your role is mapped to the **EVERYONE** special subject, then any user is allowed access. If your role is mapped to the **ALL_AUTHENTICATED** special subject, then any authenticated user is allowed access. Remove these special subjects if you want to further limit who can access your servlet or JSP. Finally, check what group the user belongs to. Although the user might not have explicit access, the group might be mapped to the role. In this case, consider removing the user from the authorized group or considering removing the group from role mapping. See Configuring authorization for applications on the Liberty profile.

Single Sign On (SSO) does not work.

If SSO does not work, check that the different Liberty profile servers that use the same LTPA keys, password, and **ssoCookieName** attribute of webAppSecurity element, each server has the same Universal Time (UTC), and the same user registry is shared. Also, if the token expires or if the cookie is sent from a web browser after changing the user registry in a manner that is inconsistent, like modifying the realm or removing the user the cookie represents, the SSO fails and the user is prompted to enter the credential information again. See Customizing SSO configuration using LTPA cookies for the Liberty profile.

Debugging security public APIs.

WSSecurityHelper and RegistryHelper are loaded even if security is not enabled. For example, if a security feature - security-1.0 or zosSecurity-1.0 - is not specified. If security is not enabled, then WSSecurityHelper.isServerSecurityEnabled() and WSSecurityHelper.isGlobalSecurityEnabled() methods both return false, and RegistryHelper.getUserRegistry() method returns null.

Other security public API classes might not be loaded if security is not enabled. If you try to access these classes you might get a java.lang.NoClassDefFoundError exception if you try to call a method on the class.

To avoid getting `java.lang.NoClassDefFoundError` exceptions, you must first test to see whether security is enabled by calling the WSSecurityHelper.isServerSecurityEnabled() or WSSecurityHelper.isGlobalSecurityEnabled() class, and then call other security public API classes only when security is enabled. See Liberty profile: Security public APIs for examples of this coding technique.

Note: This behavior is different from the full profile. In full profile, all classes are always loaded regardless of whether security is enabled or not.

Troubleshooting LDAP

This section describes some common LDAP problems and solutions you can choose.

FFDC1015I: An FFDC Incident has been created: "javax.naming.ServiceUnavailableException: myldapserver.mycompany.com:636; socket closed com.ibm.ws.security.registry.ldap.internal.LdapRegistry 298

This message in `messages.log` indicates that the configured LDAP server cannot be reached. Check your LDAP server to verify that it is running and that its IP address can be accessed from the Liberty profile server.

The javax.naming.CommunicationException: simple bind failed: myldapserver.mycompany.com:636 [Root exception is javax.net.ssl.SSLHandshakeException: com.ibm.jsse2.util.g: PKIX path building failed: java.security.cert.CertPathBuilderException: unable to find valid certification path to requested target]

If you enable SSL on your LDAP server without copying the signer of the LDAP server into the truststore referenced in the `LDAPSSLSettings` element, a connection with the LDAP server fails with an SSL handshake error. Make sure that you copy the signer of the LDAP server into your truststore.

The javax.naming.CommunicationException: myldapserver.mycompany.com:389 [Root exception is java.net.BindException: Address already in use: connect]

This message might appear in the FFDC logs and indicates that the usable sockets available on the local server are exhausted, resulting in a failure when connecting to your LDAP server. Make sure the socket is not used and try again.

CWWKS1100A: Authentication did not succeed for user ID xxxxx. An invalid user ID or password was specified

This FFDC exception could happen on the server even though the user mentioned in the message above is a valid user on LDAP server under heavy load. LDAP configuration allows to add the property `reuseConnection=false` or disable it using the developer tools. To fix the problem, Setting this property to default value of `true`.

Troubleshooting SSL

This section describes some common SSL problems and solutions you can choose.

CWWKS9105E: Could not determine the SSL port for automatic redirection.

This error occurs when a user tries to access an application that redirects to an SSL port and the SSL port is not available. The port might not be available due to a missing SSL configuration or some error in the SSL configuration definition. Check the SSL configuration in the `server.xml` file to make sure that it exists and is correct. See Securing communications with the Liberty profile.

FFDC1015I: An FFDC Incident has been created: "java.lang.IllegalArgumentException: Unknown, incomplete configuration: missing id field com.ibm.ws.config.internal.cm.ManagedServiceFactoryTracker aSyncReadNupdate. Exception thrown while trying to read configuration and update ManagedServiceFactory. Exception = java.lang.IllegalArgumentException: Unknown, incomplete configuration: missing id field" at

ffdc_12.04.18_16.09.42.0.log

This error occurs when there is a keystore element in the configuration without an ID field. If you are using a minimal SSL configuration, set the ID field to `defaultKeyStore`.

Liberty profile: Trace and logging

The product has a unified logging component. It provides base implementations of trace and FFDC services, for runtime code and for your code, to gather debug information.

The trace and FFDC implementations both apply an initial configuration during static initialization. You can modify this default initial configuration by specifying properties in the server configuration files (see Configuring the Liberty profile runtime environment) or `bootstrap.properties` file (see Specifying Liberty profile bootstrap properties).

Messages are written to stdout as well as to the defined trace destination. OSGi logging output is intercepted and output through the trace support. There is also interception of java.util.logging output.

Trace configuration

The logging service can be controlled through the server configuration. Occasionally you need to set logging properties so they can take effect before the server configuration files are processed; in this case you set them in the `bootstrap.properties` file instead of the server configuration. You do not usually need to do this to get logging from your own code, which is loaded after server configuration processing, but you might need to do this to analyze problems in early server start or configuration processing.

Logging properties can be set in either `bootstrap.properties` or `server.xml` file. You use attributes in the `server.xml` file, or use equivalent properties in the `bootstrap.properties` file. Any settings in the `bootstrap.properties` file are used when the `server.xml` file is processed. If the logging properties in the `bootstrap.properties` file are not replaced or reset in the `server.xml` file, the property values in the `bootstrap.properties` file are used.

Table 1. Logging properties for the Liberty profile. Column 1 contains attributes that can be set in the `server.xml` file. Column 2 contains equivalent properties that can be used in the `bootstrap.properties` file. Column 3 provides description of each logging property.

Attribute	Equivalent property	Description
logDirectory	com.ibm.ws.logging.log.directory	This sets the directory for all log files, including FFDC.
maxFileSize	com.ibm.ws.logging.max.file.size	The maximum size a log file is allowed to reach before being rolled (in MB). The Liberty profile runtime only does size-based log rolling. To disable this, set the value to 0. The maximum file size is approximate.
maxFiles	com.ibm.ws.logging.max.files	If there is an enforced maximum file size, this setting is used to determine how many of each log file is kept. This setting is also applied to the number of exception logs (summaries of exceptions that occurred on any given day). So if this number is 10, you might have 10 message logs, 10 trace logs, and 10 exception summaries in the `ffdc/` directory.
consoleLogLevel	com.ibm.ws.logging.console.log.level	This filter controls the granularity of messages that go to the `console.log` file. The valid values are INFO, AUDIT, WARNING, ERROR and OFF. By default, the level is AUDIT.

Table 1. Logging properties for the Liberty profile (continued). Column 1 contains attributes that can be set in the `server.xml` file. Column 2 contains equivalent properties that can be used in the `bootstrap.properties` file. Column 3 provides description of each logging property.

Attribute	Equivalent property	Description
messageFileName	com.ibm.ws.logging.message.file.name	The message log has a default name of `messagel.log`. This file always exists, and contains INFO and other (AUDIT, WARNING, ERROR, FAILURE) messages in addition to System.out and System.err. This log also contains timestamps and the issuing thread ID.
traceFileName	com.ibm.ws.logging.trace.file.name	The `trace.log` file is only created if additional or detailed trace is enabled. **stdout** is recognized as a special value, and causes trace to be directed to the original standard out stream.
traceSpecification	com.ibm.ws.logging.trace.specification	The trace string is used to selectively enabled trace. The default is *=info=enabled.
traceFormat	com.ibm.ws.logging.trace.format	This controls the format of the trace log. The default format for the Liberty profile is ENHANCED. You can also use BASIC and ADVANCED formats as in the full profile.

You can set logging properties in the server configuration file by selecting **Logging and Tracing** in the **Server Configuration** view in the developer tools, or by adding a logging element to the server configuration file like this:

```
<logging traceSpecification="*=audit=enabled:com.myco.mypackage.*=debug=enabled"/>
```

For the full logging configuration reference, including the detailed format of the **traceSpecification** property, see the `logging` element in Liberty profile: Configuration elements in the server.xml file.

Note: On all platforms, logs are written in the default system encoding.

- On z/OS systems, the default encoding is `IS08859-1`.

Liberty profile: Runtime environment known restrictions

There are a small number of known restrictions that apply when working with the Liberty profile runtime environment.

Minimum supported Java levels

The minimum supported level for the JDK from Oracle is Java 6 update 26. For the Java SDK from IBM, the minimum supported level is 6.0 (J9 2.6) SR 1.

On the z/OS platform, only 64-bit Java is supported.

For z/OS systems, use the JDK from IBM that is provided with your system.

The JDK from IBM is available in many IBM products. For example, WebSphere Application Server Version 7 includes the Java 6 JDK from IBM.

The install directory name and path cannot include non-ASCII characters

Recent JVMs do not fully support use of non-ASCII characters with the **-jar** and **-javaagent** commands. You should use only ASCII characters in your install directory names and paths.

Changing the JDBC data source at run time might result in JPA failures

If not specified through properties, OpenJPA detects and calculates the database dictionary type when the first entity manager is created and the database connection is made. This database dictionary type is used for all entity managers that are subsequently created. If the JDBC data source is changed while an application is running, the entity manager factory does not detect this change and continues to use the old dictionary for operations against the new data source. This can result in failures if the database is changed to a different vendor.

When you change a database to a different vendor, restart the application.

Modifying the dataSource, jdbcDriver, connectionManager and JDBC vendor properties at run time might result in JPA failures

If you update the configuration of `dataSource`, `jdbcDriver`, `connectionManager` or any of the JDBC vendor properties lists (for example, `properties.db2.jcc` or `properties.oracle`) while the server is running, you might see `J2CA8040E` failures. These failures state that multiple `dataSource` elements cannot be associated with a single `connectionManager`. These failures are generated even if your configuration associates only one `connectionManager` with the `dataSource` element.

When you make updates to the configuration of any of these JDBC resources, restart the server.

An application that relies on a result being returned by `getRealPath` must be deployed as an expanded application, not as a WAR file

The Java EE specification states that the `getRealPath()` method returns a `null` value if the content is being made available from a web archive (WAR) file. When you deploy a WAR file to the Liberty profile, the profile does not automatically extract the archive file into a directory structure. Therefore the application might fail to start. If your application relies on a result being returned by `getRealPath()`, you must deploy the application as an expanded web application, not as a WAR file. For example, you can manually extract the WAR file and copy the expanded application to the `dropins` directory.

WebSphere Application Server full profile scripts do not work with the Liberty profile

You cannot use any of the scripts under the `bin` directory of the WebSphere Application Server full profile to administer the Liberty profile.

Fileset restrictions

The following restrictions apply to Filesets:

- Filesets do not recursively explore subdirectories of the base directory. For example the following instructions are not supported:

```
<fileset id="testFileset" dir="\temp" includes="**\a.jar"/>
<fileset id="testFileset" dir="\temp" includes="a\a.jar"/>
<fileset id="testFileset" dir="\temp" includes="*\a.jar"/>
<fileset id="testFileset" dir="\temp" includes="a\b\a.jar"/>
```

- If you use symbolic links with Filesets, you must add a forward slash ("/") at the end of the `dir` attribute value. For example:

```
<fileset dir="${ihc.home}/" includes="*.jar"/>
```

Version 1.11 or later is required for the Liberty profile on z/OS

To use the native z/OS capabilities of the Liberty profile, you must be using z/OS Version 1.11 or later.

z/OS APAR OA37620 is required to enable server tracing for a specific trace group

If you want to trace individual trace groups, apply z/OS APAR OA37620. When enabling tracing for the Liberty profile on the z/OS platform, use the package/class syntax (that is, `com.ibm.ws.*=all`) unless directed otherwise by IBM support.

beanvalidation-1.0 feature restrictions

For the `beanvalidation-1.0` feature, the following restriction applies:

- There is no support for bean validation inside OSGi applications.

jaxrs-1.1 feature restriction

For the `jaxrs-1.1` feature, Liberty profile does not support JAX-RS third-party APIs in the developer tools as in the full profile.

jpa-2.0 feature restrictions

For the `jpa-2.0` feature, the following restrictions apply:

- There is no support for dynamic removal from the runtime.
- You cannot use Java 7 to compile and build your JPA applications. Applications that are built with Java 6 will run on the Liberty profile with Java 7.

jsf-2.0 feature restrictions

For the `jsf-2.0` feature, the following restriction applies:

- There is no support for dynamic removal from the runtime.

jsp-2.2 feature restrictions

For the `jsp-2.2` feature, the following restriction applies:

- There is no support for the `useInMemory` configuration option to only store the translated JSP file in memory.

Liberty profile: Messages

When you use the Liberty profile, you might encounter system messages. Each message has a unique message identifier, and includes an explanation of the problem, and details of any action that you can take to resolve the problem.

Liberty profile system messages are logged from a variety of sources, including application server components and applications. For the Liberty profile, the message identifier is 10 characters in length and has the following form:

```
CWXXX9999X
```

where:

CWXXX
is a five-character alphabetic message prefix that identifies the Liberty profile component.

9999 is a four-character numeric identifier used to identify the specific message for that component.

X is an optional alphabetic indicator that identifies the message type. (I=Informational, W=Warning, E=Error)

The Troubleshooter reference: Messages topic contains information about all WebSphere Application Server messages, indexed by message prefix. For each message there is an explanation of the problem, and details of any action that you can take to resolve the problem.

Table 2. Alphabetic message prefixes for Liberty profile messages

Liberty profile message prefix	Range	Liberty profile component
CWLIB	0001-0100	Transactions
CWLIB	0101-0200	z/OS Transaction Extensions
CWWKB		*z/OS Native integration*
CWWKB	0001-0050	z/OS Command Processing
CWWKB	0051-0100	z/OS Native code only
CWWKB	0101-0150	z/OS Core
CWWKB	0151-0200	z/OS WLM services
CWWKC	0000-0250	Annotation scanning
CWWKE		*Core kernel and kernel services*
CWWKE	0001-0099	Boot/Launcher
CWWKE	0100-0199	Service utilities
CWWKE	0200-0299	Location service
CWWKE	0300-0399	File install service
CWWKE	0400-0499	FileMonitor service
CWWKE	0500-0599	Extensible Class Scanner
CWWKF		Feature manager
CWWKG		Configuration manager
CWWKJ		Light touch administration
CWWKL	0001-0100	Class loading service
CWWKM	0001-0050	Artifact API messages (container factory)
CWWKM	0051-0100	Overlay Messages (all implementations)
CWWKM	0101-0150	Artifact API Zip implementation
CWWKM	0151-0200	Artifact API File implementation
CWWKM	0401-0450	Adaptable API Implementation messages. (adaptable module factory / adapter service)

Table 2. Alphabetic message prefixes for Liberty profile messages (continued)

Liberty profile message prefix	Range	Liberty profile component
CWWKM	1001-1100	Loose archive API (used by the Eclipse-based tools option that runs an application directly from the Eclipse workspace(The Eclipse platform supports a "virtual" application archive, where a set of files that appear to be in the same archive file are actually spread out across the Eclipse workspace. Liberty calls this a “loose archive”, and the tools option that uses the loose archive API is called **Run this application directly from the workspace**)).
CWWKM	2000-2999	Ant and maven plugin
CWWKN	0001-0100	JNDI default namespace
CWWKO		*CFW components*
CWWKO	0000-0199	CFW
CWWKO	0200-0399	TCP
CWWKO	0400-0599	UDP
CWWKO	0600-0799	bytebuffer
CWWKO	0800-0899	SSL channel
CWWKO	0900-0999	SSL
CWWKS		*Security*
CWWKS	*0000 series*	*Security: General messages*
CWWKS	0000-0099	Security
CWWKS	0900-0999	Security quickStart
CWWKS	*1000 series*	*Security: Authentication services*
CWWKS	1000-1099	Authentication
CWWKS	1100-1199	JAAS authentication
CWWKS	1200-1299	TAI authentication
CWWKS	*2000 series*	*Security: Authorization services*
CWWKS	2000-2099	Authorization
CWWKS	2100-2199	Built-in authorization
CWWKS	2900-2999	SAF authorization
CWWKS	*3000 series*	*Security: Registry services*
CWWKS	3000-3099	Registry
CWWKS	3100-3199	Basic registry
CWWKS	3200-3299	LDAP registry
CWWKS	3300-3399	FileBased (VMM) registry
CWWKS	3800-3899	Custom registry
CWWKS	3900-3999	SAF registry
CWWKS	*4000 series*	*Security: Token services*
CWWKS	4000-4099	Token services

Table 2. Alphabetic message prefixes for Liberty profile messages (continued)

Liberty profile message prefix	Range	Liberty profile component
CWWKS	4100-4199	LTPA
CWWKS	4200-4299	Kerberos
CWWKS	4300-4399	SPNEGO
CWWKS	*9000 series*	*Security: Collaborators*
CWWKS	9100-9199	Web Collaborator (common code)
CWWKS	9200-9299	Web Application Collaborator
CWWKS	9300-9399	Web Administration Collaborator
CWWKT		HTTP transport/dispatcher
CWWKX		*JMX*
CWWKX	0001-0100	JMX Security
CWWKX	0101-0200	JMX REST Connector
CWWKX	0201-0300	JMX REST Client
CWWKZ		*Applications*
CWWKZ	0001-0100	Application manager
CWWKZ	0101-0200	WARs
CWWKZ	0201-0300	WABs
CWWKZ	0301-0400	EBAs

Liberty profile: Abend reason codes

When you use the Liberty profile, you might encounter abend reason codes. Each message has a unique message identifier, and includes an explanation, and details of any action that you can take.

Note: Abend (reason) codes not listed in this chapter should always be directly reported to the IBM Support Center.

Table 3. Liberty profile for z/OS abend codes

Abend Code	Abend Reason	Explanation	Suggested Action
EC3 hex	20000005	The BBGZSAFM module could not be loaded from the file system.	Check that the installation of the Liberty profile completed without errors, and that the user ID which is running the Liberty server has permission to read the files in the Liberty installation. If the problem persists, contact your next level of support.
EC3 hex	20000006	The BBGZSAFM module is not APF authorized.	Check that the installation of the Liberty profile completed without errors. The BBGZSAFM module should have been installed with the APF authorization bit set. If the problem persists, contact your next level of support.

Table 3. Liberty profile for z/OS abend codes (continued)

Abend Code	Abend Reason	Explanation	Suggested Action
EC3 hex	20000007	A call to Unix System Services stat function to the BBGZSAFM load module failed.	Check that the installation of the Liberty profile completed without errors, and that the user ID which is running the Liberty server has permission to read the files in the Liberty installation. If the problem persists, contact your next level of support.
EC3 hex	20000008	The path name to the BBGZSAFM module in the file system is too long. The path should not be more than 4096 characters.	Install the Liberty profile to location in the file system which can be accessed with a shorter PATH.
EC3 hex	20000009	The size of the BBGZSAFM module is smaller than the minimum possible size for this load module.	Check that the installation of the Liberty profile completed without errors. If the problem persists, contact your next level of support.
EC3 hex	2000000A	The size of the BBGZSAFM module is smaller than the minimum possible size for this load module based on the number of functions it contains.	Check that the installation of the Liberty profile completed without errors. If the problem persists, contact your next level of support.

Chapter 3. Debugging applications

To debug your application, you must use a development environment like the IBM Rational® Application Developer for WebSphere to create a Java project. You must then import the program that you want to debug into the project.

About this task

By following the steps below, you can import the WebSphere Application Server examples into a Java project. Two debugging styles are available:

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

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

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

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

Procedure

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

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

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

     When you have added this JAR file, select it and use the **Attach Source** function to attach the `Increment.jar` file because it contains both the source and class files.
   - *app_server_root*`/dev/JavaEE/j2ee.jar`
   - *app_server_root*`/plugins/com.ibm.ws.runtime.jar`
   - *app_server_root*`/plugins/com.ibm.ws.webcontainer.jar`

   Click **OK** when you have added all of the JARs.
9. You can set some breakpoints in the source at this time if you like, however, it is not necessary as step-by-step mode will prompt you whenever the server calls a method on a web object. Step-by-step mode is explained in more detail below.

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

Attaching a Rational tool to a remote debug session

The steps below describe how to attach an IBM Rational Application Developer for WebSphere Software product to a remote debug session on WebSphere Application Server for z/OS.

About this task

Remote debugging can prove useful when the program you are debugging behaves differently on a z/OS system than on your local system.

Procedure

1. Enable the debug engine on WebSphere Application Server for z/OS using the administrative console. See debugging service details.
2. Import the Java source code that you want to debug into a Rational Application Developer product and set breakpoints. See topics on setting breakpoints in the Rational Application Developer documentation.
3. In the Rational Application Developer product, open a debug perspective and create a debug session configuration.
4. Attach the Rational Application Developer product to the WebSphere Application Server for z/OS debug runtime. See topics on remote debugging in the Rational Application Developer documentation.
5. Run the Java code in WebSphere Application Server for z/OS to hit the breakpoints set in the Rational Application Developer product.
6. Use the debugger controls and features to debug the application.

Unit testing with DB2

These steps describe how to setup a unit test environment that would allow you to develop and unit test code with DB2® z/OS to support Container Managed Persistence (CMP) development and access DB2 test data that resides on z/OS.

About this task

When using DB2 z/OS to support Container Managed Persistence (CMP) development and access DB2 test data that resides on z/OS, you should establish a testing environment to develop and unit test the code. Perform the steps below to setup a test environment:

Procedure

1. Configure DB2 Distributed Data Facility (DDF) on z/OS to allow remote TCP/IP connections from your WebSphere Studio Application Developer workstation. See the DB2 Information Center for information on DDF.

2. Install the DB2 Client Configuration Assistant on the workstation where WebSphere Studio Application Developer is installed. The DB2 Client Configuration Assistant is shipped with DB2.
3. Use the DB2 Client Configuration Assistant to define a DB2 alias.
4. Use the DB2 alias you defined to access the DB2 subsystem on z/OS using the DB2 Distributed Data Facility (DDF).

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** > **Servers Types** > **WebSphere application servers** > ***server name*** > **Debugging service**.

You can enable a debug session on WebSphere Application Server on this page. 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.

Chapter 4. Adding logging and tracing to your application

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

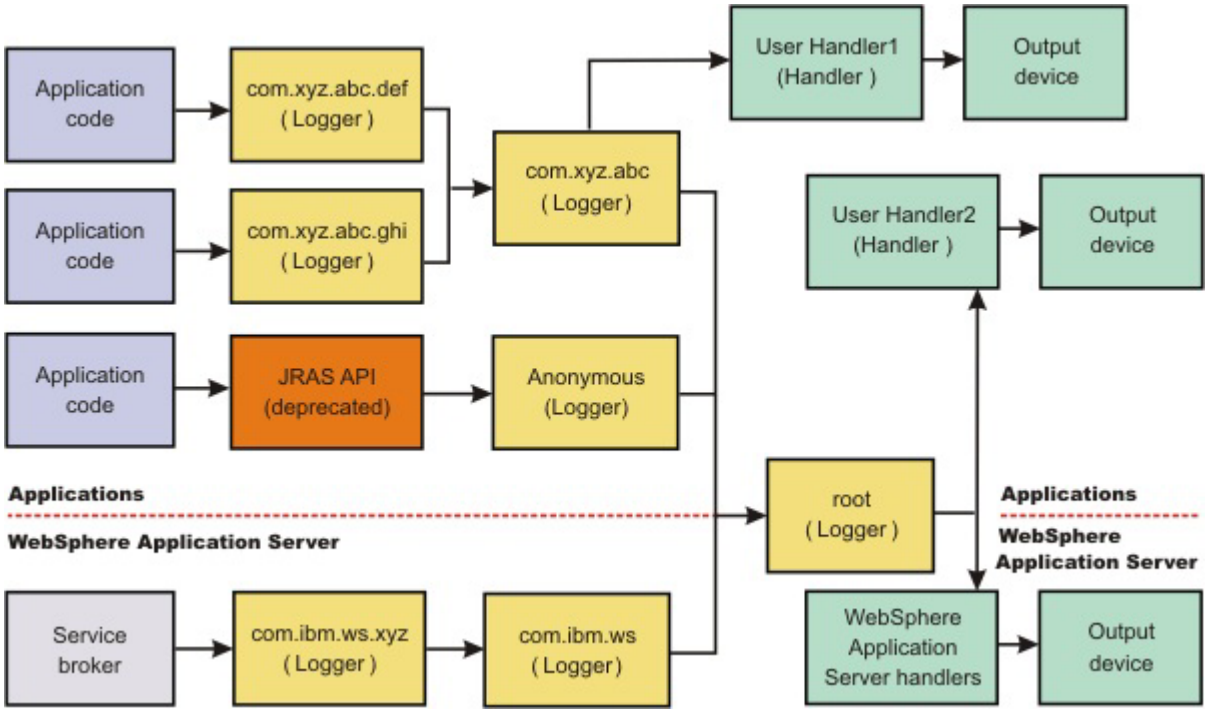
About this task

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

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

This approach has advantages over adding `System.out.println` statements to your code:

- Your messages are displayed in the WebSphere Application Server standard log files, using a standard message format with additional data, such as a date and time stamp that are added automatically.
- You can more easily correlate problems and events in your own application to problems and events that are associated with WebSphere Application Server components.
- You can take advantage of the WebSphere Application Server log file management features.



Procedure

1. Enable and configure any of the supported types of logging as needed. Use one of the following methods:
   - Configuring Java logging using the administrative console
   - Configuring applications to use Jakarta Commons Logging
2. Customize the properties to meet your logging needs. For example, enable or disable a particular log, specify the number of logs to be kept, and specify a format for log output.
   - Configuring Java logging using the administrative console

3. If you do not want log and trace from Jakarta Commons Logging to use the WebSphere log and trace infrastructure, reconfigure the Jakarta Commons Logging.
   - "Configuring applications to use Jakarta Commons Logging" on page 35

   Note: Use the WebSphere log and trace infrastructure for all of your log content to make problem source identification simpler.
4. Restart the application server after making static configuration changes.

Example

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

Project name	Location
EJB projects	ejbModule/META-INF/MANIFEST.MF
Application client projects	appClientModule/META-INF/MANIFEST.MF
Dynamic web projects	WebContent/META-INF/MANIFEST.MF
Connector projects	connectorModule/META-INF/MANIFEST.MF

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

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

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

Using Java logging in an application

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

About this task

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

Procedure

1. Optional: Create the necessary handler, formatter, and filter classes if you need your own log files.

   Note: Use the WebSphere log and trace infrastructure to make problem source identification simpler, rather than creating separate log files.
2. Optional: If localized messages are used by the application, create a resource bundle, as described in "Creating log resource bundles and message files" on page 32.
3. In the application code, get a reference to a logger instance, as described in "Using a logger" on page 21.

4. Insert the appropriate message and trace logging statements in the application, as described in “Using a logger.”

Using a logger

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

About this task

Java provides a log and trace package, java.util.logging, that you can use to instrument your applications. This topic provides recommendations about how to use the log and trace package.

Procedure

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

   For messages use:

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

   For trace use:

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

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

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

Localized messages

The following sample applies to localized messages:

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

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

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

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

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

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


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

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

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

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

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

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

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


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

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

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

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

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

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

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


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

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

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

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

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

if (logger.isLoggable(Level.INFO))
 logger.logrb(Level.INFO, className, methodName, resourceBundleNameSpecial,
```

```
"MSG_KEY_01", "parameter 1");

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

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

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

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

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

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

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

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

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


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


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

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

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


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

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

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


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

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

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


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

There may be occasions when you want to propagate log records to your own log handlers rather than participate in integrated logging. To use a stand-alone log handler, set the useParentHandlers flag to `false` in your application.The mechanism for creating a customer handler is the Handler class support that is provided by the IBM Developer Kit, Java Technology Edition. If you are not familiar with handlers, as implemented by the Developer Kit, you can get more information from various texts, or by reading the API documentation for the java.util.logging API. The following sample shows a custom handler:

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

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

 FileOutputStream fileOutputStream;
 PrintWriter printWriter;

 public MyCustomHandler(String filename) {
  super();

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

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

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

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

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

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

A custom filter provides optional, secondary control over what is logged, beyond the control that is provided by the level. The mechanism for creating a custom filter is the Filter interface support that is provided by the IBM Developer Kit, Java Technology Edition. If you are not familiar with filters, as implemented by the Developer Kit, you can get more information from various texts, or by reading the API documentation the for the java.util.logging API.

The following example shows a custom filter:

```
/**
 * This class filters out all log messages starting with SECJ022E, SECJ0373E, or SECJ0350E.
 */
import java.util.logging.Filter;
import java.util.logging.Handler;
import java.util.logging.Logger;
import java.util.logging.LogRecord;

public class MyFilter implements Filter {
 public boolean isLoggable(LogRecord lr) {
  String msg = lr.getMessage();
  if (msg.startsWith("SECJ0222E") || msg.startsWith("SECJ0373E") || msg.startsWith("SECJ0350E")) {
   return false;
  }
  return true;
 }
}

//This code will register the above log filter with the root Logger's handlers (including the WAS system logs):
...
Logger rootLogger = Logger.getLogger("");
rootLogger.setFilter(new MyFilter());
```

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

The following example shows a custom formatter:

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

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

 public MyCustomFormatter() {
  super();
 }

 public String format(LogRecord record) {

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

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

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

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

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

Adding custom handlers, filters, and formatters enables you to customize your logging environment beyond what can be achieved by the configuration of the default WebSphere Application Server logging infrastructure. The following example demonstrates how to add a new handler to process requests to the com.myCompany subtree of loggers (see “Configuring the logger hierarchy” on page 31). The main method in this sample gives an example of how to use the newly configured logger.

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

public class MyCustomLogging {

 public MyCustomLogging() {
  super();
 }

 public static void initializeLogging() {

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

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

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

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

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

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

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

 }

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

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

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

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

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

Loggers

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

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

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

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

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

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

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

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

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

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

The logger provides several sets of methods for generating log messages. Some log methods take only a level and enough information to construct a message. Other, more complex logp (log precise) methods support the caller in passing class name and method name attributes, in addition to the level and message information. The logrb (log with resource bundle) methods add the capability of specifying a resource bundle as well as the level, message information, class name, and method name. Using methods such as severe, warning, fine, finer, and finest you can log a message at a particular level. For more information on logging and how to use it in your applications read “Using Java logging in an application” on page 20. For a complete list of methods, see the java.util.logging documentation at http://java.sun.com/javase/.

Log handlers

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

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

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

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

Handlers can have a formatter. If a formatter is supplied, the formatter controls how the logged objects are formatted. For example, the formatter can decide to first include the time stamp, followed by a string representation of the level, followed by the message that is included in the logged object. The handler writes this formatted representation to the output device.

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

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Log levels

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

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

The level value is set from configuration data when the logger is created and can be changed at run time from the administrative console. If a level is not set in the configuration data, a level is obtained by proceeding up the hierarchy until a parent with a level value is found. You can also set a level for each

handler to indicate which events are published to an output device. When you change the level for a logger in the administrative console, the change is propagated to the children of the logger.

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

Table 4. Valid log levels. This table lists valid logging levels.

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
Finer	Trace information - Detailed trace + method entry / exit / return values
Finest	Trace information - A more detailed trace - Includes all the detail that is needed to debug problems
All	All events are logged. If you create custom levels, All includes your custom levels, and can provide a more detailed trace than Finest.

For instructions on how to set logging levels, read the topic about configuring Java logging using the administrative console.

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

Log filters

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

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

A filter is attached to a logger or log handler using the appropriate setFilter method. For a complete list of filter methods, see the java.util.logging documentation at http://java.sun.com/javase/

Log formatters

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

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

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

Formatters are attached to handlers using the setFormatter method.

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

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Java logging

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

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

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

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

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

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

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

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the

LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Note: The `SystemOut.log` and `STDOUT` streams are redirected to the `SYSPRINT` ddname under z/OS. The `SystemErr.log` and `STDERR` streams are redirected to the `SYSOUT` ddname under z/OS. By default, the WebSphere Application Server for z/OS cataloged procedures associate these ddnames with print (SYSOUT=*) data sets, causing message logs to go into WebSphere Application Server job output. Job output can be viewed with the Spool Display and Search Facility (SDSF) or equivalent software.

Configuring the logger hierarchy

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

About this task

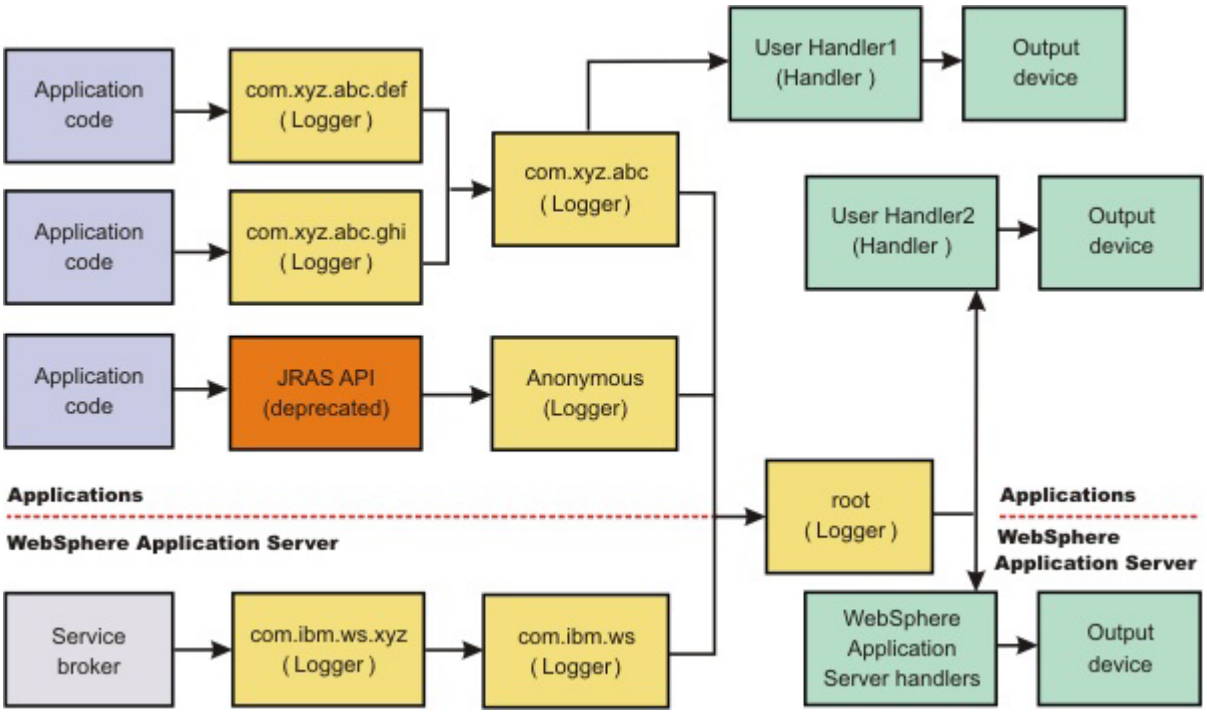
You can configure your application server to handle logs in many different ways. Configure your log settings based upon your configuration and the logging structure that best suits your needs.

Procedure

- Forward all application logging requests to the WebSphere Application Server handlers. This behavior is the default.
- Forward all application logging requests to your own custom handlers. Set the **useParentHandlers** option to `false` on one of your custom loggers, and then attach your handlers to that logger.
- Forward all application logging requests to both WebSphere Application Server handlers, and your custom handlers, but do not forward WebSphere Application Server logging requests to your custom handlers. Set the **useParentHandlers** option to `true` on one of your non-root custom loggers, and then attach your handlers to that logger.`True` is the default setting.
- Forward all WebSphere Application Server logging requests to both WebSphere Application Server handlers, and your custom handlers. 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.

Example

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



Creating log resource bundles and message files

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

About this task

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

The PropertyResourceBundle class is the preferred mechanism to use.

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

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

the localized text in the resource bundle and formats the message. Use the early binding technique to package the application resource bundles with the application.

To create a resource bundle, perform the following steps.

Procedure

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

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

Example

You can create resource bundles in several ways. The best and easiest way is to create a properties file that supports a properties resource bundle. This example 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 are inserted. All the normal properties file conventions and rules apply to this file. In addition, the creator must be aware of other restrictions that are imposed on the values by the Java MessageFormat class. For example, apostrophes must be escaped or they cause a problem. Avoid the use of non-portable characters. WebSphere Application Server does not support the use of extended formatting conventions that the MessageFormat class supports, such as {1, date} or {0,number, integer}.

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

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

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

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

What to do next

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

Logger.properties file for configuring logger settings

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

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

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

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

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

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

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

Syntax of the Logger.properties file

Use the following syntax to set logger properties:

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

where:

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

property is one of the following properties:

- organization
- product
- component
- minimum_localization_level
- group
- eventfactory
- handler_preference=operator (This property writes anything that is logged to the console WTO, write-to-operator. Without this property the AUDIT level is written only to hardcopy WTO.)

Sample Logger.properties file

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

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

Group Logger.properties file

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

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

Configuring applications to use Jakarta Commons Logging

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

Before you begin

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

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

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

To better understand Jakarta Commons Logging, read Jakarta Commons and the specifications for Java Logging and for Common Base Event. To better understand use of the WebSphere Application Server logger, read “Jakarta Commons Logging” on page 36.

About this task

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

best-practices: The default configuration of Jakarta Commons Logging is stored in the `commons-logging.properties` file. To specify the factory class to use with Jakarta Commons Logging in an application, provide a file named `org.apache.commons.logging.LogFactory`, located in `META-INF/services` directory, that contains the name of the factory class on the first line. This is the configuration mechanism for the JAR file service provider, as defined in JDK 1.3 and above.

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

Procedure

1. Examine “Configurations for the WebSphere Application Server logger” on page 39 and determine which configuration best suits your application.
2. Change your application configuration as needed to enable use of the WebSphere Application Server logger.

Results

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

Jakarta Commons Logging

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

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

Support for Jakarta Commons Logging

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

The product support for Jakarta Commons Logging does not change interfaces defined by Jakarta Commons Logging.

Benefits of support for Jakarta Commons Logging

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

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

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

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

Overview of the process for using Jakarta Commons Logging

Logging with Jakarta Commons Logging consists of the steps that follow. “Configurations for the WebSphere Application Server logger” on page 39 provides details on configuring your application to use the WebSphere Application Server logger.

1. Obtain an instance of a logger factory.

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

2. Obtain an instance of a logger.

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

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

   CommonBaseEvent
   Wrapped into CommonBaseEventLogRecord

   CommonBaseEventLogRecord
   Passed without change

   LogRecord
   Passed without change

   Other objects
   Converted to String

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

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

Classes used to obtain a logger factory and logger

Table 5. Jakarta Commons Logging class descriptions. Use the classes for a logger factory instance and logger.

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

Table 5. Jakarta Commons Logging class descriptions (continued). Use the classes for a logger factory instance and logger.

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

Logger level configuration and mapping

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

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

Table 6. Mapping of WsJDK14Logger levels to Java Logging levels. Compare the logging levels.

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

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

Configurations for the WebSphere Application Server logger

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

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

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

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

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

Table 7. Conditions required to use logger. When Parent First and Jakarta Commons Logging is bundled with an application.

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

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

Table 8. Conditions required to use logger. When Parent First and Jakarta Commons Logging is not bundled with an application.

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

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

Table 9. Conditions required to use logger. When Parent Last and Jakarta Commons Logging is bundled with an application.

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

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

Table 10. Conditions required to use logger. When Parent Last and Jakarta Commons Logging is not bundled with an application.

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

Programming with the JRas framework

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

Before you begin

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

About this task

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

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Procedure

1. Retrieve a reference to the JRas manager.
2. Retrieve message and trace loggers by using methods on the returned manager.
3. Call the appropriate methods on the returned message and trace loggers to create message and trace entries, as appropriate.

JRas logging toolkit

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

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

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

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

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

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

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

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

JRas concepts

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

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

Event classes

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

Loggers

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

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

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

Handlers

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

Formatters

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

JRas Extensions

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

JRas extensions

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

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

environment. The collection of these implementation classes is referred to as the *JRas extensions*.

Usage model

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

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

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

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

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

JRas extension classes

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

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

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

WebSphere Application Server message and trace loggers

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

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

Loggers are named objects and are identified by name. When the Manager class is called to obtain a logger, the caller is required to specify a name for the logger. The Manager class maintains a

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

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

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

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

WebSphere Application Server handlers

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

WebSphere Application Server formatters

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

WebSphere Application Server manager

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

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

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

JRas framework (deprecated)

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

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

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

Handlers

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

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

Formatters

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

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

Message event types

The stand-alone JRas toolkit defines message event types in the RASIMessageEvent interface. In addition, the WebSphere Application Server reserves a range of message event types for future use. The RASIMessageEvent interface defines three types, with values of `0x01`, `0x02`, and `0x04`. The values `0x08` through `0x8000` are reserved for future use. You can provide your own message event types by extending this interface appropriately. User-defined message types must have a value of `0x1000` or greater.

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

Message event objects

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

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

Trace event types

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

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

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

Trace event objects

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

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

User defined types, user defined events and WebSphere Application Server

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

JRas programming interfaces for logging (deprecated):

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

General considerations

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

Writing a handler

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

RASIObject interface

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

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

RASIMaskChangeGenerator interface

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

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

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

- The public void addMaskChangeListener(RASIMaskChangeListener listener) and public void removeMaskChangeListener(RASIMaskChangeListener listener) methods are used to add or remove

listeners to the handler. The public Enumeration getMaskChangeListeners method returns an enumeration over the list of currently registered listeners. The public void fireMaskChangedEvent(RASMaskChangeEvent mc) method is used to call back all the registered listeners to inform them of a mask change event.

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

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

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

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

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

RASIHandler interface

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

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

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

Writing a formatter

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

RASIFormatter interface

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

Programming model summary

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

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

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

- You must not use implementation classes that are provided by the stand-alone JRas logging toolkit directly, unless specifically noted otherwise. Direct usage of those classes is not supported. IBM Support provides no diagnostic aid or bug fixes relating to the direct use of classes that are provided by the stand-alone JRas logging toolkit.
- You must obtain message and trace loggers directly from the Manager class. You cannot directly instantiate loggers.
- You cannot replace the WebSphere Application Server message and trace logger classes.
- You must guarantee that the logger names that are passed to the Manager class are unique, and follow the documented naming constraints. When a logger is obtained from the Manager class, you must not attempt to change the name of the logger by calling the setName method.
- Named loggers can be used more than once. For any given name, the first call to the Manager class results in the Manager class creating a logger that is associated with that name. Subsequent calls to the Manager class that specify the same name result in a returned reference to the existing logger.
- The Manager class maintains a hierarchical namespace for loggers. Use a dot-separated, fully qualified class name to identify any logger. Other than dots or periods, logger names cannot contain any punctuation characters, such as an asterisk (*), a comma (.), an equals sign (=), a colon (:), or quotes.
- Group names must comply with the same naming restrictions as logger names.
- The loggers returned from the Manager class are subclasses of the RASMessageLogger and the RASTraceLogger classes that are provided by the stand-alone JRas logging toolkit. You can call any public method that is defined by the RASMessageLogger and RASTraceLogger classes. You cannot call any public method that is introduced by the provided subclasses.
- If you want to operate in either stand-alone or combined mode, you must provide your own Handler and Formatter subclasses. You cannot use the Handler and Formatter classes that are provided by the stand-alone JRas logging toolkit. User written handlers and formatters must conform to the documented guidelines.

- Loggers that are obtained from the Manager class come with a WebSphere Application Server handler installed. This handler writes message and trace records to logs that are defined by the WebSphere Application Server run time. Manage these logs using the provided systems management interfaces.
- You can programmatically add and remove user-defined handlers from a logger at any time. Multiple additions and removals of user defined handlers are supported. You are responsible for creating an instance of the handler to add, configuring the handler by setting the handler mask value and formatter appropriately, then adding the handler to the logger using the addHandler method. You are responsible for programmatically updating the masks of user-defined handlers, as appropriate.
- You might get a reference to the handler that is installed within a logger by calling the getHandlers method on the logger and processing the results. You must not call any methods on the handler that are obtained in this way. You can remove the WebSphere Application Server handler from the logger by calling the logger removeHandler method, passing in the reference to the WebSphere Application Server handler. When removed, the WebSphere Application Server handler cannot be added again to the logger.
- You can define your own message type. The behavior of user-defined message types and restrictions on their definitions is discussed in JRas framework (deprecated).
- You can define your own message event classes. The use of user-defined message event classes is discussed in JRas framework (deprecated).
- You can define your own trace types. The behavior of user-defined trace types and restrictions on your definitions is discussed in JRas framework (deprecated).
- You can define your own trace event classes. The use of user-defined trace event classes is discussed in JRas framework (deprecated).
- You must programmatically maintain the bits in the message and trace logger masks that correspond to any user-defined types. If WebSphere Application Server facilities are used to manage the predefined types, these updates must not modify the state of any of the bits that correspond to those types. If you are assuming ownership responsibility for the predefined types, then you can change all bits of the masks.

JRas messages and trace event types

The basic JRas message and event types are not the same as those natively recognized by WebSphere Application Server, so the JRas types are mapped onto the types that are native to the runtime environment. You can control the way JRas message and trace events are processed using custom filters and message controls.

Event types

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

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

Platform message event types

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

Table 11. Platform message event types. 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

Table 11. Platform message event types (continued). These message types are mapped onto the native message types, as follows.

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

Application developers can use JRas to issue an MVS™ WTO (write to operator) message by using a JRas RASIMessageEvent type of TYPE_INFO or TYPE_INFORMATION to issue a WebSphere Application Server for z/OS Audit trace. A WebSphere Application Server for z/OS Audit trace maps to an MVS route code 11 WTO (hardcopy WTO).

Platform trace event types

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

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

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

Table 12. WebSphere Application Server native types and JRas RASITraceEvent level types. Mapping WebSphere Application Server trace types to the JRas RASITraceEvent level types.

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

Table 13. WebSphere Application Server native types and JRas RASITraceEvent enumerated types. Mapping WebSphere Application Server trace types to the JRas RASITraceEvent enumerated types.

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

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

Message and trace parameters

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

Primitives

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

Object

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

Object[]

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

Throwable

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

Array of primitives

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

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

Controlling message logging

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

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

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

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

Control tracing

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

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

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

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

Instrumenting an application with JRas extensions

You can create an application using JRas extensions.

Before you begin

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

About this task

To create an application using the WebSphere Application Server JRas extensions, perform the following steps:

Procedure

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

Creating JRas resource bundles and message files

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

Before you begin

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

About this task

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

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

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

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

The two techniques are described as follows:

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

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

To create a resource bundle, perform the following steps.

Procedure

1. Create a text properties file that lists message keys and the corresponding messages. The properties file must have the following characteristics:
   - Each property in the file is terminated with a line-termination character.

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

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

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

What to do next

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

JRas resource bundles:

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

Resource bundle sample

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

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

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

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

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

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

JRas manager and logger instances

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

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

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

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

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

Using the message and trace loggers

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

Using a message logger

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

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

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

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

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

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

Using a trace logger

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

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

Setting up for integrated JRas operation

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

Before you begin

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

About this task

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

Procedure

1. Import the requisite JRas extensions classes:

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

2. Declare logger references:

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

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

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

   // Configure the message logger to use the message file that is created
   // for this application.
   ```

```
msgLogger.setMessageFile("acme.widgets.DefaultMessages");
trcLogger = mgr.createRASTraceLogger("Acme", "Widgets", "RasTest",
         myTestBean.class.getName());
mgr.addLoggerToGroup(trcLogger, groupName);
```

Setting up for combined JRas operation

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

Before you begin

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

About this task

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

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

Procedure

1. Import the requisite JRas extensions classes:

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

2. Import the user handler and formatter:

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

3. Declare the logger references:

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

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

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

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

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

Setting up for stand-alone JRas operation

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

Before you begin

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

About this task

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

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

Procedure

1. Import the requisite JRas extensions classes:

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

2. Import the user handler and formatter:

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

3. Declare the logger references:

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

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

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

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

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

      // Add the Handler to the logger. Add the logger to the list of the
      // handlers listeners, then set the handlers
      // mask, which will update the loggers composite mask appropriately.
   ```

```
// WARNING - there is an order dependency here that must be followed.
msgLogger.addHandler(handler);
handler.addMaskChangeListener(msgLogger);
handler.setMessageMask(RASIMessageEvent.DEFAULT_MESSAGE_MASK);
```

Logging messages and trace data for Java server applications

By using the WebSphere Application Server for z/OS support for logging application messages and trace data, you can improve the reliability, availability, and serviceability of any Java application that runs in a WebSphere Application Server for z/OS server.

About this task

Through this support, your Java application's messages can appear on the MVS master console, in the error log stream, or in the component trace (CTRACE) data set for WebSphere Application Server for z/OS. Your application's trace entries can appear in the same CTRACE data set.

Procedure

1. Determine where to issue the log messages. Read “Message location best practices” for tips on which tools to use.
2. Configure logging in the MVS master console, the error log stream, or the CTRACE data set.

Message location best practices

Use this information for configuring messaging locations.

You might want to issue messages to the MVS master console to report serious error conditions for mission-critical applications. Through the master console, an operator can receive and, if necessary, take action in response to a message that indicates the status of an application. In addition, by directing messages to the master console, you can trigger automation packages to take action for specific conditions or events related to your application's processing.

Any messages that your application issues to the console also appear in either the error log stream or the CTRACE data set for WebSphere Application Server for z/OS, depending on the message type. Logging the messages in these system resources can help you more easily diagnose errors related to your application's processing. Similarly, issuing requests to log trace data in the CTRACE data set is another method of recording error conditions or collecting application data for diagnostic purposes.

Automation-geared messages

Table 14. Messages that can help with automation. The table lists the messages that can help with automation.

Message ID	Message text
BBOO0001I	WEBSPHERE FOR Z/OS CONTROL PROCESS %s/%s/%s/%s IS STARTING.
BBOO0002I	WEBSPHERE FOR Z/OS CONTROL PROCESS %s ENDED NORMALLY. .
BBOO0003E	WEBSPHERE FOR Z/OS CONTROL PROCESS %s ENDED ABNORMALLY, REASON=%X.
BBOO0004I	WEBSPHERE FOR Z/OS SERVANT PROCESS %s/%s/%s/%s IS STARTING.
BBOO0005I	WEBSPHERE FOR Z/OS SERVANT PROCESS %s ENDED NORMALLY.
BBOO0006E	WEBSPHERE FOR Z/OS PROCESS %s ENDED ABNORMALLY, REASON=%X.
BBOO0007I	WEBSPHERE FOR Z/OS DAEMON %s/%s/%s/%s IS STARTING.
BBOO0008I	WEBSPHERE FOR Z/OS DAEMON %s ENDED NORMALLY.
BBOO0009E	WEBSPHERE FOR Z/OS DAEMON %s ENDED ABNORMALLY, REASON=%X.
BBOO0015I	INITIALIZATION COMPLETE FOR DAEMON %s.

Table 14. Messages that can help with automation (continued). The table lists the messages that can help with automation.

Message ID	Message text
BBOO0019I	INITIALIZATION COMPLETE FOR WEBSPHERE FOR Z/OS CONTROL PROCESS %s.
BBOO0020I	INITIALIZATION COMPLETE FOR WEBSPHERE FOR Z/OS SERVANT PROCESS %s.
BBOO0035W	TERMINATING THE CURRENT PROCESS, REASON=%08X.
BBOO0048W	WEBSPHERE FOR Z/OS COMPONENT TRACE MAY HAVE LOST ENTRIES.
BBOO0057W	AUTOMATIC RESTART MANAGER (IXCARM) SERVICE '%s' WARNING/FAILURE, RC=%X REASON=%X.
BBOO0093E	WEBSPHERE FOR Z/OS DAEMON NOT FOUND BY SERVER %s.
BBOO0095E	WEBSPHERE FOR Z/OS DAEMON FAILED TO INITIALIZE BECAUSE ANOTHER DAEMON WAS STARTING.
BBOO0100E	WEBSPHERE FOR Z/OS CONTROL PROCESS %s FAILED TO INITIALIZE BECAUSE ANOTHER CONTROL PROCESS OF THE SAME NAME WAS STARTING
BBOO0101E	WEBSPHERE FOR Z/OS CONTROL PROCESS %s FAILED TO INITIALIZE BECAUSE ANOTHER CONTROL PROCESS OF THE SAME NAME WAS ACTIVE.
BBOO0102W	WEBSPHERE FOR Z/OS UNEXPECTED DELAY WAITING FOR START OF SERVER %s.
BBOO0103W	WEBSPHERE FOR Z/OS WAITING FOR START OF SERVER %s, SECOND WARNING.
BBOO0104E	WEBSPHERE FOR Z/OS WAITING FOR START OF SERVER %s HAS TIMED OUT.
BBOO0128E	WEBSPHERE FOR Z/OS UNABLE TO LOCATE RRS BBOO0131W ERROR ENCOUNTERED DURING DAEMON ATTEMPT TO STOP SERVER %s.
BBOO0136E	UNABLE TO START SERVER %s BECAUSE WEBSPHERE FOR Z/OS DAEMON IS STOPPING
BBOO0138W	WEBSPHERE FOR Z/OS UNEXPECTED DELAY WAITING FOR STOP OF SERVER %s.
BBOO0139W	WEBSPHERE FOR Z/OS WAITING FOR STOP OF SERVER %s, SECOND WARNING.
BBOO0140E	WEBSPHERE FOR Z/OS WAITING FOR STOP OF SERVER %s HAS TIMED OUT.
BBOO0144I	ARM DETECTED A FAILURE AND IS RESTARTING THIS SERVER
BBOO0145E .	ARM REGISTRATION FAILED - ARM COUPLE DATASET FULL
BBOO0146I	ARM REGISTRATION FAILED - ARM DETECTED A DUPLICATE NAME
BBOO0147I	ARM READY FAILED - ARM TIMEOUT EXCEEDED
BBOO0150E	COMMAND IGNORED, STOP COMMAND ALREADY ISSUED FOR SERVER %s
BBOO0151E	COMMAND IGNORED, MODIFY CANCEL COMMAND ALREADY ISSUED FOR SERVER %s
BBOO0165E	ARM REGISTRATION FAILED - TIMEOUT
BBOO0172I	WEBSPHERE FOR Z/OS SERVANT PROCESS %s NOT STARTING ON CONFIGURED SYSTEM %s
BBOO0173I	SERVER %s/%s ACTIVE ON %s AT LEVEL %s%s
BBOO0228E	WEBSPHERE FOR Z/OS SERVER FAILED BECAUSE DAEMON GROUP %s IS NOT ACTIVE.
BBOO0236I	UNIX SYSTEM SERVICES SHUTDOWN INITIATED. ISSUING STOP TO DAEMON %s.
BBOO0237I	WEBSPHERE FOR Z/OS DAEMON %s/%s/%s IS STARTING.
BBOO0238I	WEBSPHERE FOR Z/OS CONTROL PROCESS %s/%s/%s IS STARTING
BBOO0239I	WEBSPHERE FOR Z/OS SERVANT PROCESS %s/%s/%s IS STARTING.
BBOO0242E	%s STARTUP IS DELAYED, WAITING FOR INFORMATION FROM GRS.
BBOO0246I	INITIALIZATION COMPLETE FOR DAEMON %s/%s/%s/%s.

Table 14. Messages that can help with automation (continued). The table lists the messages that can help with automation.

Message ID	Message text
BBOO0247I	INITIALIZATION COMPLETE FOR WEBSPHERE FOR Z/OS CONTROL PROCESS %s/%s/%s/%s.
BBOO0248I	INITIALIZATION COMPLETE FOR WEBSPHERE FOR Z/OS SERVANT PROCESS %s/%s/%s/%s.
BBOO0277I	WEBSPHERE FOR Z/OS MULTI-PRODUCT PTF POST INSTALLER STARTING.
BBOO0278I	WEBSPHERE FOR Z/OS MULTI-PRODUCT PTF POST INSTALLER ENDED NORMALLY.
BBOO0279W	POST INSTALLER ENCOUNTERED WARNING(S) WHILE APPLYING SERVICE. REPLY 'CONTINUE' OR 'CANCEL' '
BBOO0285A	CONFIGURED ROOT SERVICE LEVEL CHECK FAILED REPLY 'CONTINUE' OR 'CANCEL'
BBOO0286A	BACKWARDS INCOMPATIBLE POST INSTALL ACTION(S) PENDING. NOTE FOR UNINSTALL. REPLY 'CONTINUE' OR 'CANCEL
BBOO0287A	SERVER IS STARTING OUT OF PLACE AT MIXED PTF LEVELS. REPLY 'CONTINUE' OR 'CANCEL'
BBOO0288I	WEBSPHERE FOR Z/OS ADJUNCT PROCESS %s/%s/%s/%s IS STARTING.
BBOO0289I	WEBSPHERE FOR Z/OS ADJUNCT PROCESS %s ENDED NORMALLY.
BBOO0290E	WEBSPHERE FOR Z/OS ADJUNCT PROCESS %s ENDED ABNORMALLY, REASON=%X.
BBOO0291I	INITIALIZATION COMPLETE FOR WEBSPHERE FOR Z/OS ADJUNCT PROCESS %s.
BBOO0292I	INITIALIZATION COMPLETE FOR WEBSPHERE FOR Z/OS ADJUNCT PROCESS %s/%s/%s/%s.
BBOO0293I	WEBSPHERE FOR Z/OS ADJUNCT PROCESS %s/%s/%s IS STARTING.
BBOO0294I	WEBSPHERE FOR Z/OS CONTROL PROCESS %s RESTART ISSUED, RC =%X.
BBOO0297A	SERVER %s/%s/%s HAD NO SERVANTS AND IS REJECTING WORK. REPLY 'CONTINUE' TO ACCEPT WORK.
BBOO0298E	SERVER %s/%s/%s IS CURRENTLY MODIFYING COMMUNICATION LISTENERS. THE %s MODIFY OPTION IS NOT ALLOWED AT THIS TIME.
BBOO0299I	SERVER %s/%s/%s HAS NO SERVANTS. WORK IS BEING REJECTED.
BBOO0300I	SERVER %s/%s/%s HAS DETECTED SERVANT(S). WORK IS NO LONGER BEING REJECTED.
BBOO0301E	SERVER %s/%s/%s HAS NO SERVANTS. SERVER HAS FAILED A NUMBER OF ATTEMPTS TO MODIFY ITSELF TO REJECT WORK.
BBOO0323I	MAXIMUM CONFIGURED NUMBER OF CONNECTIONS %d FOR HOSTNAME/IP: %s PORT: %d REACHED. NEW CONNECTIONS MAY NOT BE ACCEPTED.

System performance when logging messages and trace data

Using message logging and trace data can affect system performance depending on your system configuration.

You can select the amount and types of trace data to be collected, which provides you with the ability to either run your application with minimal tracing when performance is a priority, or run your application with detailed tracing when you need to recreate a problem and collect additional diagnostic information.

The error log stream, the CTRACE data set for WebSphere Application Server for z/OS, and the master console are primarily intended for monitoring or recording diagnostic data for system components and critical applications. Depending on your installation's configuration, directing application messages and

data to these resources might have an adverse affect on system performance. For example, if you send application data to the CTRACE data set, trace entries in that data set might wrap more quickly, which means you might lose some critical diagnostic data because the system writes new entries over existing ones when wrapping occurs. Use this logging support judiciously.

Note: You can only use WebSphere Application Server for z/OS support for logging messages and trace data for Java applications, not for Java applets.

Issuing application messages in the MVS master console

With the WebSphere Application Server for z/OS reliability, availability, and serviceability support for Java (JRas) framework, you can issue messages from your Java application to the MVS master console. You might want to issue messages to the master console to report serious error conditions for mission-critical applications, or to trigger automation packages.

Before you begin

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

The messages your application issues also appear in either the error log stream or the component trace (CTRACE) data set that WebSphere Application Server for z/OS uses.

Logging the messages is another method of recording error conditions or collecting application data for diagnostic purposes.

About this task

WebSphere Application Server for z/OS provides code that creates and manages a message logger, which processes your application's messages. WebSphere Application Server for z/OS creates only one message logger for each unique organization, product, or component, so that you can more easily identify the messages recorded in the error log stream or CTRACE data set for a specific application. The message logger runs in the Java virtual machine (JVM) for the WebSphere Application Server for z/OS server in which your Java application will run.

To use a message logger, in your Java application:

Procedure

1. Log messages and trace data for Java server applications.
2. Drive the method to instruct WebSphere Application Server for z/OS to create the message logger.
3. Code messages at appropriate points in your application.

Logging Common Base Events in WebSphere Application Server

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

About this task

Attention: Logging Common Base Events is not supported with the High Performance Extensible Logging (HPEL) log and trace mode.

An *event* is a notification from an application or the application server that reports information that is related to a specific problem or situation. Common Base Events provide you with a standard structure for these event notifications, which allow you to correlate events that are received from different applications.

Log Common Base Events to capture events from different sources to help you fix a problem within an application environment or to tune system performance.

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.

Procedure

1. Optional: Read about the Common Base Event types and how they are implemented within an application server. Refer to "The Common Base Event in WebSphere Application Server."
2. Read "Logging Common Base Events in WebSphere Application Server" on page 89.
3. Configure the Common Base Event framework for your application server using one of the following methods:
   - "Logging with Common Base Event API and the Java logging API" on page 78
   - "Generate Common Base Event content with the default event factory" on page 80.

Results

Common Base Events will now be logged according to your configuration. Use these event logs to determine the source of application problems.

The Common Base Event in WebSphere Application Server

The Common Base Event is an XML document that defines a common representation of events that is intended for use by enterprise management and business applications. The Common Base Event defines common fields, the values they can take, and the exact meanings of these values.

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

The application that creates the event object is called the *event source*. Event sources can use a common structure for the event. The accepted standard for such a structure is called the *Common Base Event*. The Common Base Event is an XML document that is defined as part of the autonomic computing initiative.

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

Note:

For WebSphere Application Server Version 8.5, if you delete an application server that was previously deployed with the Common Event Infrastructure (CEI) enabled and you did not uninstall CEI before deleting the server, you must use a different name when creating an application server

that you want to deploy with CEI. If you deploy CEI on an application server that was created with the exact same server name as the server that was previously deleted and CEI was not uninstalled, the following error occurs:

```
com.ibm.websphere.management.exception.AdminException: ADMA5026E: No valid target is specified in ObjectName
WebSphere:cell=targetCell,node=targetNode,server=targetServer for module EventServerMdb.jar+META-INF/ejb-jar.xml
```

If you did not uninstall CEI before deleting the application server, you must ensure that you use a name for the new application server that is different from the name of the server that was previously deployed with the common event infrastructure.

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

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

For complete details on the Common Base Event format, see the XML schema that is included in the Common Base Event specification document, at http://www.ibm.com/developerworks/autonomic/books/fpy0mst.htm#HDRCBEDESC .

Types of problem determination events

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

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

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

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

Common Base Event structure

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

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

- Common header information
- Component identification, both source and reporter
- Situation information

- Message data
- Extended data
- Context data
- Associated events and association engine

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

The following table presents a summary of all the fields in the Common Base Event and their usage requirements for problem determination events.

Table 15. Field name, log events, and base specification. This table shows whether a particular element or attribute is required, recommended, optional, prohibited, or discouraged for log events, and the base specification.

Field name	Log events	Base specification
Version	Required	Required
creationTime	Required	Required
severity	Required	Optional
Msg	Required	Optional
sourceComponentId*	Required	Required
sourceComponentId.location	Required	Required
sourceComponentId.locationType	Required	Required
sourceComponentId.component	Required	Required
sourceComponentId.subComponent	Required	Required
sourceComponentId.componentIdType	Required	Required
sourceComponentId.componentType	Required	Required
sourceComponentId.application	Recommended	Optional
sourceComponentId.instanceId	Recommended	Optional
sourceComponentId.processId	Recommended	Optional
sourceComponentId.threadId	Recommended	Optional
sourceComponentId.executionEnvironment	Optional	Optional
situation*	Required	Required
situation.categoryName	Required	Required
situation.situationType*	Required	Required
situation.situationType.reasoningScope	Required	Required
situation.situationType.(specific Situation Type elements)	Required	Required
msgDataElement*	Recommended	Optional
msgDataElement .msgId	Recommended	Optional
msgDataElement .msgIdType	Recommended	Optional
msgDataElement .msgCatalogId	Recommended	Optional
msgDataElement .msgCatalogTokens	Recommended	Optional
msgDataElement .msgCatalog	Recommended	Optional
msgDataElement .msgCatalogType	Recommended	Optional
msgDataElement .msgLocale	Recommended	Optional
extensionName	Recommended	Optional
localInstanceId	Optional	Optional
globalInstanceId	Optional	Optional

Table 15. Field name, log events, and base specification (continued). This table shows whether a particular element or attribute is required, recommended, optional, prohibited, or discouraged for log events, and the base specification.

Field name	Log events	Base specification
priority	Discouraged	Optional
repeatCount	Optional	Optional
elapsedTime	Optional	Optional
sequenceNumber	Optional	Optional
reporterComponentId*	Optional	Optional
reporterComponentId.location	Required (2)	Required (2)
reporterComponentId.locationType	Required (2)	Required (2)
reporterComponentId.component	Required (2)	Required (2)
reporterComponentId.subComponent	Required (2)	Required (2)
reporterComponentId.componentIdType	Required (2)	Required (2)
reporterComponentId.componentType	Required (2)	Required (2)
reporterComponentId.instanceId	Optional	Optional
reporterComponentId.processId	Optional	Optional
reporterComponentId.threadId	Optional	Optional
reporterComponentId.application	Optional	Optional
reporterComponentId.executionEnvironment	Optional	Optional
extendedDataElements*	Note 3	Optional
contextDataElements*	Note 4	Optional
associatedEvents*	Note 5	Optional

Notes:

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

Common header information:

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

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

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

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

severity

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

Table 16. Severity values. The following values are used for problem determination events:

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

Table 16. Severity values (continued). The following values are used for problem determination events:

Value	Severity	Description
50	Critical	Critical errors describe events that represent an unrecoverable error within a component. The failure significantly affects the component ability to service most requests. The business application can continue most, but not all of its normal functions and its overall operation might be degraded. These events require administrator action or intervention to address the condition.
60	Fatal	Fatal errors describe events that represent an unrecoverable error within a component. The failure usually results in the complete failure of the component. The business application can continue some normal functions, but its overall operation might be degraded. These events require administrator action or intervention to address the condition.

msg
Refer to “Message data” on page 74 for information on this attribute.

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

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

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

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

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

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

Component identification for source and reporter:

The component identification fields in the Common Base Event are used to indicate which component in the system is experiencing the condition that is described by the event (the sourceComponentID) and which component emitted the event (the reporterComponentID).

Typically, these components are the same, in which case only the sourceComponentID is supplied. Some notes and scenarios on when to use these two elements in the Common Base Event:

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

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

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

Table 17. Component identification for source and reporter. The information that is used to identify a component in the system is the same, regardless of whether it is the source component or reporter component.

Source component	Reporter component	Description
location locationType	Component location	Identifies the location of the component.
component componentIdType	Component name	Identifies the asset name of the component, as well as the type of component.
subcomponent	Subcomponent name	Identifies a specific part or subcomponent of a component, for example a software module or hardware part.
application	Business application name	Identifies the business application or process the component is a part of and provides services for.
instanceId	Operational instance	Identifies the operational instance of a component, that is the actual running instance of the component.
processId threadId	Operational instance	Identifies the operational instance of a component within the context of a software operating system, that is he operating system process and thread running when the event was produced.

Table 17. Component identification for source and reporter (continued). The information that is used to identify a component in the system is the same, regardless of whether it is the source component or reporter component.

Source component	Reporter component	Description
executionEnvironment	Operational instance Component location	Provides additional information about the operational instance of a component or its location by identifying the name of the environment hosting the operational instance of the component, for example the operating system name for a software application, the application server name for a Java 2 Platform, Enterprise Edition (J2EE) application, or the hardware server type for a hardware part.

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

Component

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

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

subComponent

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

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

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

componentIdType

The componentIdType field is required by the Common Base Event specification, but provides minimal value for problem determination events. For most problem determination events, it is encouraged to use the value provided in the application field instead of the componentIdType. The componentIdType field identifies the type of component; the application is identified by the application field.

application

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

field is not required for problem determination events is that instances exist where the issuing component might not be aware of the overall business application.

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

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

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

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

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

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

executionEnvironment

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

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

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

Situation information:

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

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

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

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

Message data:

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

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

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

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

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

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

msg
All message, log, and trace events must provide a human-readable message in the msg field of the

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

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

msgLocale

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

msgId and msgIdType

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

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

msgCatalogId

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

msgCatalogTokens

The msgCatalogTokens field is listed as an optional value within the Common Base Event specification, but provide this value whenever the Common Base Event includes localized or translated message text, for example when providing problem determination events that represent issued messages or log events. This field is not required for problem determination events because not all problem determination events include translated message text, and cases exist where the value is not provided or available when formatting the Common Base Event. This value contains the list of locale-independent values or message tokens that are inserted into the localized message text when creating a translated message.

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

The Common Base Event provides several mechanisms for providing additional data about an event, including this field, extended data elements, and extensions to the schema. Always use the

msgCatalogTokens field to supply the list of message tokens that is included in the message text associated with an event. These values can also be supplied in other parts of the Common Base Event, but they must be included in this field.

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

Extended data:

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

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

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

Sample Common Base Event instance

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

Use the following example for reference:

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

 ... several extendedDataElements for internal use only ...

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

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

 <situation categoryName="ReportSituation">
```

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

</CommonBaseEvent>
```

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

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

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

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

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

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

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

Sample Common Base Event template

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

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

The following example illustrates a Common Base Event template:

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

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

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

```
  </extendedDataElements>
 </CommonBaseEvent>

</TemplateEvent>
```

Component identification for problem determination

This topic describes types of problem determination events.

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

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

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

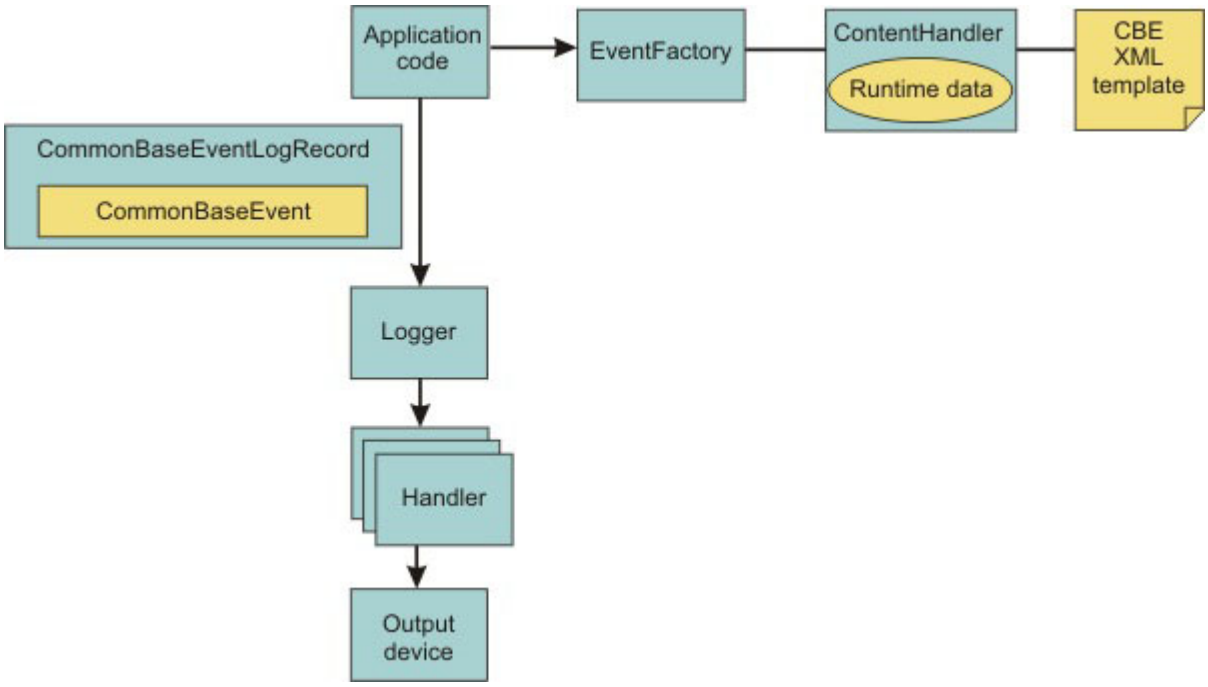
Subcomponents
: A specific component, depending on its complexity, might consist of several subcomponents. For example, the IBM WebSphere Application Server consists of many subcomponents, such as the enterprise bean container and the servlet engine. Subcomponent information is typically used only by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

Logging with Common Base Event API and the Java logging API

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

Before you begin

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



About this task

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

Common Base Events are initiated and logged in the following sequence:

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

Procedure

- You can use the default Common Base Event factory to generate content. Read “Generate Common Base Event content with the default event factory” on page 80 for more information.
- If you do not wish to use the default event factory, you can create custom content handlers and event factories.
  1. Create a custom factory home. Read “Creating custom Common Base Event factory homes” on page 84.

2. Create a custom content handler. Read “Creating custom Common Base Event content handlers” on page 83.

Results

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

Generate Common Base Event content with the default event factory

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

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

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

To specify a Common Base Event template in the previous case, a Logger.properties file would need to be provided with an eventfactory entry for *com.ibm.someLogger*. If a valid template is found on the classpath, then the Logger's event factory will use the specified template's content in addition to the WebSphere Application Server runtime information when populating Common Base Events. If the template is not found on the classpath, or is invalid, then the Logger's event factory will only use the WebSphere Application Server runtime information when populating Common Base Events.

The default content handler is also associated with the event factory home supplied in the global event factory context. This is convenient for creating Common Base Events that need to be populated with content similar to that generated from the WebSphere Application Server:

```
// Request the event factory from the global event factory home
EventFactory eventFactory =
   EventFactoryContext.getInstance().getEventFactoryHome().getEventFactory(templateName);

// Create a Common Base Event
CommonBaseEvent commonBaseEvent = eventFactory.createCommonBaseEvent();

// Complete the Common Base Event using content from the template (if specified previously)
// and the server runtime information.
eventFactory.getContentHandler().completeEvent(commonBaseEvent);
```

In the previous example, if the template referenced by *templateName* is found on the classpath, and the template is valid, then the event factory home will return an event factory which uses a content handler that combines the template's content with the WebSphere Application Server runtime information when populating Common Base Events. If the template is not found on the classpath, or is invalid, then the event factory home will return an event factory which uses a content handler that uses only the WebSphere Application Server runtime information when populating Common Base Events.

The default content handler populates Common Base Events in the server environment with the following runtime information:

CommonBaseEvent.globalInstanceId

Value: The unique_record_id

Set this value only if the CommonBaseEvent.globalInstanceId value is null before the completeEvent method is called.

CommonBaseEvent.msg
Value: A localized message that is based on the MsgDataElement element.

Set this value only if the CommonBaseEvent.msg message is null before the completeEvent method is called.

CommonBaseEvent.severity
Value: Set based on the value of level set on the CommonBaseEventLogRecord record, if level >= Level.SEVERE, set to `50`; if level >= Level.WARNING, set to `30`; the default is set to `10`.

Set this value only if the CommonBaseEvent.severity value is null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.component
Value: Set based on the LoggerName value that is set on the CommonBaseEventLogRecord record.

Set this value only if the CommonBaseEvent.ComponentIdentification.component is null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.componentIdType
Value: “Unknown”

Set this value only if the CommonBaseEvent.ComponentIdentification.componentIdType value is null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.executionEnvironment
Value: `OSname[OSarch]#OSversion`

Set this value only if the CommonBaseEvent.ComponentIdentification.executionEnvironment value is null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.instanceId
Value: `cellName\nodeName\serverName`

Set this value only if the CommonBaseEvent.ComponentIdentification.instanceId value is null before the completeEvent method is called. Set only in a server environment because this value is ignored in a client application.

CommonBaseEvent.ComponentIdentification.location
Value:The host name

Set this value only if both the CommonBaseEvent.ComponentIdentification.location and the CommonBaseEvent.ComponentIdentification.locationType values are null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.locationType
Value: The host name

Set this value only if both the CommonBaseEvent.ComponentIdentification.location and the CommonBaseEvent.ComponentIdentification.locationType values are null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.processId
Value: An internally generated representation of the process number.

Set this value only if the CommonBaseEvent.ComponentIdentification.processId value is null before the completeEvent method is called

CommonBaseEvent.ComponentIdentification.subComponent
Value: Set based on values of the sourceClassName and the sourceMethodName names that are set on the sourceClassName.sourceMethodName name of the CommonBaseEventLogRecord record.

Set this value only if the CommonBaseEvent.ComponentIdentification.subComponent values is null before the completeEvent method is called and both the sourceClassName and the sourceMethodName names are set.

CommonBaseEvent.ComponentIdentification.threadId
Value: Set to the value of the Java Virtual Machine (JVM) thread name.

Set this value only if the CommonBaseEvent.ComponentIdentification.threadId values is null before the completeEvent value is called.

CommonBaseEvent.ComponentIdentification.componentType
Value: http://www.ibm.com/namespaces/autonomic/WebSphereApplicationServer

Set this value only if the CommonBaseEvent.ComponentIdentification.componentType values is null before the completeEvent method is called.

CommonBaseEvent.MsgDataElement.msgLocale
Value: Set based on the default locale of the JVM.

Set this value only if the CommonBaseEvent.msg value is null before the completeEvent method is called.

CommonBaseEvent.Situation.categoryName
Value: `ReportSituation`

Set this value only if the CommonBaseEvent.Situation value is null before the completeEvent method is called.

CommonBaseEvent.Situation.situationType.type
Value: `ReportSituation`

Set this value only if the CommonBaseEvent.Situation value is null before the completeEvent method is called.

CommonBaseEvent.Situation.situationType.reasoningScope
Value: EXTERNAL

Set this value only if the CommonBaseEvent.Situation value is null before the completeEvent method is called.

CommonBaseEvent.Situation.situationType.reportCategory
Value: LOG

Set this value only if the CommonBaseEvent.Situation value is null before the completeEvent method is called.

The sourceComponentIdentification value is populated if no reporterComponentIdentification ID exists when the completeEvent method is invoked on the content handler. Otherwise, the reporterComponentIdentification ID is populated instead.

Common Base Event content handler

Content handlers populate data into Common Base Events when the Common Base Event complete method is invoked. You can associate content handlers with Common Base Event templates, which provide default information to transfer into each Common Base Event.

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

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

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

Creating custom Common Base Event content handlers

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

Before you begin

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

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

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

About this task

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

Procedure

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

```
public class CustomContentHandler extends WsContentHandlerImpl {

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

 public void completeEvent(CommonBaseEvent cbe) throws CompletionException {
  // following code will add WAS content to the Content Base Event
```

```
  super.completeEvent(cbe);
  // TODO Custom content can be added to the Content Base Event here
 }
}
```

- The following shows how to implement the CustomTemplateContentHandler class:

```
public class CustomTemplateContentHandler extends WsTemplateContentHandlerImpl {

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

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

Results

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

Common Base Event factory home

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

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

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

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

Creating custom Common Base Event factory homes

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

Before you begin

Event factory homes create and provide homes for Event Factory instances. Each event factory home has a content handler. This content handler is assigned to every event factory the event factory home creates. In turn, when a Common Base Event is created, the content handler from the event factory is assigned to it. Event factory instances are maintained by the associated event factory home, based on their unique name. For example, when application code requests a named event factory, the newly created event factory instance is returned and persisted for future requests for that named event factory.

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

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

About this task

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

Procedure

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

```
public class CustomEventFactoryHome extends AbstractEventFactoryHome {

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

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

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

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

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

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

1. Implement the CustomTemplateEventFactoryHome class by using this code:

```
public class CustomTemplateEventFactoryHome extends
  EventXMLFileEventFactoryHomeImpl {

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

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

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

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

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

- The CustomEventFactoryHelper class can be implemented and used by following the code below:

1. Implement the custom CustomEventFactoryHelper class using this code:

```
public class CustomTemplateEventFactoryHome extends
  EventXMLFileEventFactoryHomeImpl {

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

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

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

Figure 4 CustomTemplateEventFactoryHome class

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

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

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

```
      break;
     }
     return factory;
    }

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

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

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

Results

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

Common Base Event factory context

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

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

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

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

Common Base Event factory

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

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

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

java.util.logging -- Java logging programming interface

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

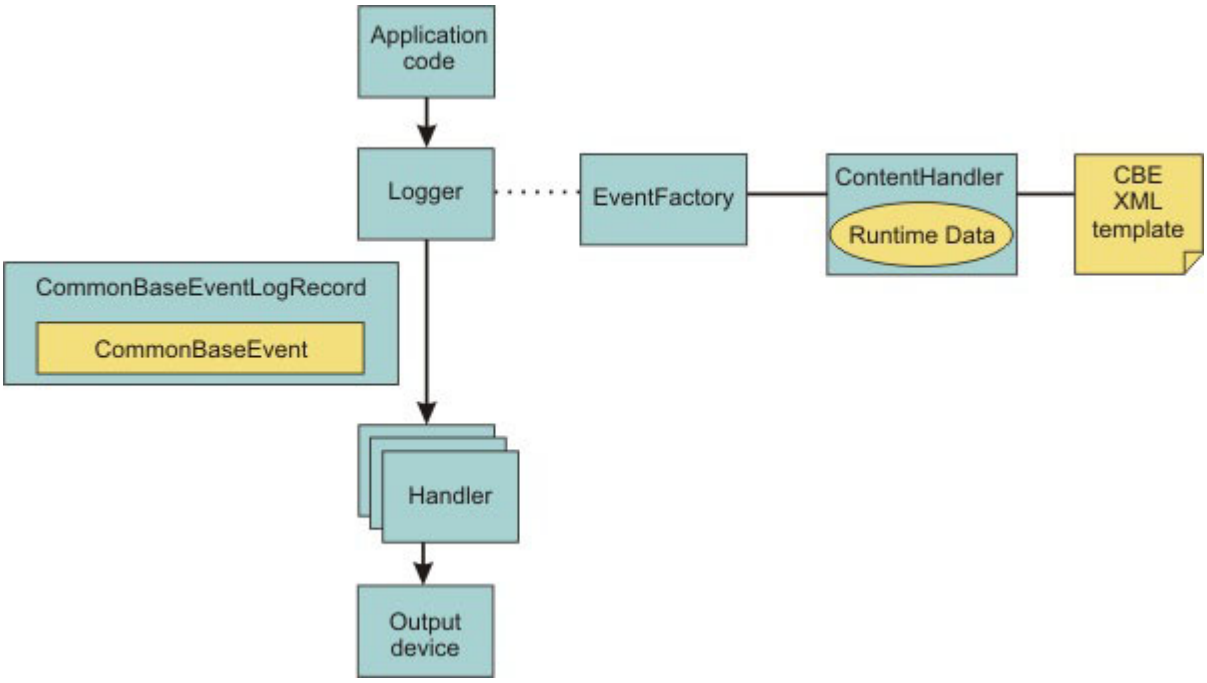
In the WebSphere Application Server, when using basic log and trace mode, the Java logging API (java.util.logging) automatically creates Common Base Events for events that are logged at the WsLevel.DETAIL level or later (including WsLevel.DETAIL, Level.CONFIG, Level.INFO, WsLevel.AUDIT, Level.WARNING,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

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

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

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

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



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

1. Application code invokes the named logger (WsLevel.DETAIL or later) with event-specific data.
2. The logger creates a Common Base Event using the createCommonBaseEvent method on the event factory that is associated with the logger.

3. The logger creates a Common Base Event using the event factory associated to the logger.
4. The logger wraps the common base event in a CommonBaseEventLogRecord record, and adds event-specific data.
5. The logger calls the Common Base Event complete method.
6. The Common Base Event invokes the ContentHandler completeEvent method.
7. The content handler adds XML template data to the Common Base Event (including for example, the component name). Not all content handlers support templates.
8. The content handler adds runtime data to the Common Base Event (including for example, the current thread name).
9. The logger passes the CommonBaseEventLogRecord record to the handlers.
10. The handlers format data and write to the output device.

Logger.properties file

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

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

The logging subsystem uses Common Base Events to represent all the messages in the WebSphere Application Server `activity.log` file. You can specify your own event factory template to be used with your loggers. Use the eventfactory property in your `Logger.properties` file. See “Sample Common Base Event template” on page 77 for details on the Common Base Event template.

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

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

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

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

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

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

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

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Logging Common Base Events in WebSphere Application Server

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

Follow these guidelines:

- Use a different logger for each component. Sharing loggers across components gets in the way of associating loggers with component-specific information.
- Associate loggers with event templates that specify source component identification. This association ensures that the source of all events created with the logger is properly identified.
- Use the same template for directly created Common Base Events (events created using the Common Base Event factories) and indirectly created Common Base Events (events created using the Java logging API) within the same component.
- Avoid calling the complete method on Common Base Events until you are finished adding data to the Common Base Event and are ready to log it. This approach ensures that any decisions made by the content handler based on data already in the event are made using the final data.

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

```
com.ibm.componentX.eventfactory=com.ibm.componentX
```

The following sample code demonstrates the use of the same event factory setting for direct (Part 1) and indirect (Part 2) Common Base Event logging:

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

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

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

< /TemplateEvent>
```

Showlog commands for Common Base Events

The **showlog** command converts the service log from binary format into plain text.

Purpose

These **showlog** commands to produce output in Common Base Event XML format.

- `showlog -start` *startDateTime* `-format CBE-XML-1.0.1` *logStreamName*

where:

startDateTime
: Specifies the start date and time, in yyyy-MM-ddTHH:mm:ss.SSSZ format. Milliseconds and time zone are optional.

logStreamName
: Is the name of the configured error log stream.

For examples of showlog scripts, see Viewing the service log.

Chapter 5. Configuring Java logging using the administrative console

Java logging provides a standard logging API for your applications. 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.

About this task

Developing, deploying and maintaining applications are complex tasks. When an application encounters an unexpected condition, it might not be able to complete a requested operation. You might want the application to inform the administrator that the operation failed and tell the administrator why the operation failed. This information enables the administrator to take the proper corrective action. Application developers might need to gather detailed information that relates to the path of a running application to determine the root cause of a failure that is due to a code bug. The facilities that are used for these purposes are typically referred to as *logging* and *tracing*. For more information read “Java logging” on page 30.

Using the administrative console, you can:

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

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

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

Procedure

1. Set the logging levels for your logs:
   a. In the navigation pane, click **Servers > Server Types > WebSphere application servers**.
   b. Click the name of the server that you want to work with.
   c. Under Troubleshooting, click **Logs and Trace**.
   d. Click **Change Log Detail levels**.
   e. To make a static change to the configuration, click the **Configuration** tab. A list of well-known components, packages, and groups is displayed. To change the configuration dynamically, click the **Runtime** tab. The list of components, packages, and groups displays all the components that are currently registered on the running server.
   f. Select a component, package, or group to set a logging level.
   g. [High Performance Extensible Logging] Select whether or not you want to disable the logging and tracing of potentially sensitive data.
   h. Click **Apply**.
   i. Click **OK**.
2. To have static configuration changes take effect, stop then restart the application server.

Log streams and expected output

Investigating the logging and tracing output for the application server is an excellent way to observe performance, diagnose problems, and gain a general understanding of how the application server is working within your environment. The expected output locations for logging and trace information can be different depending on the operating system on which the application server is running.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

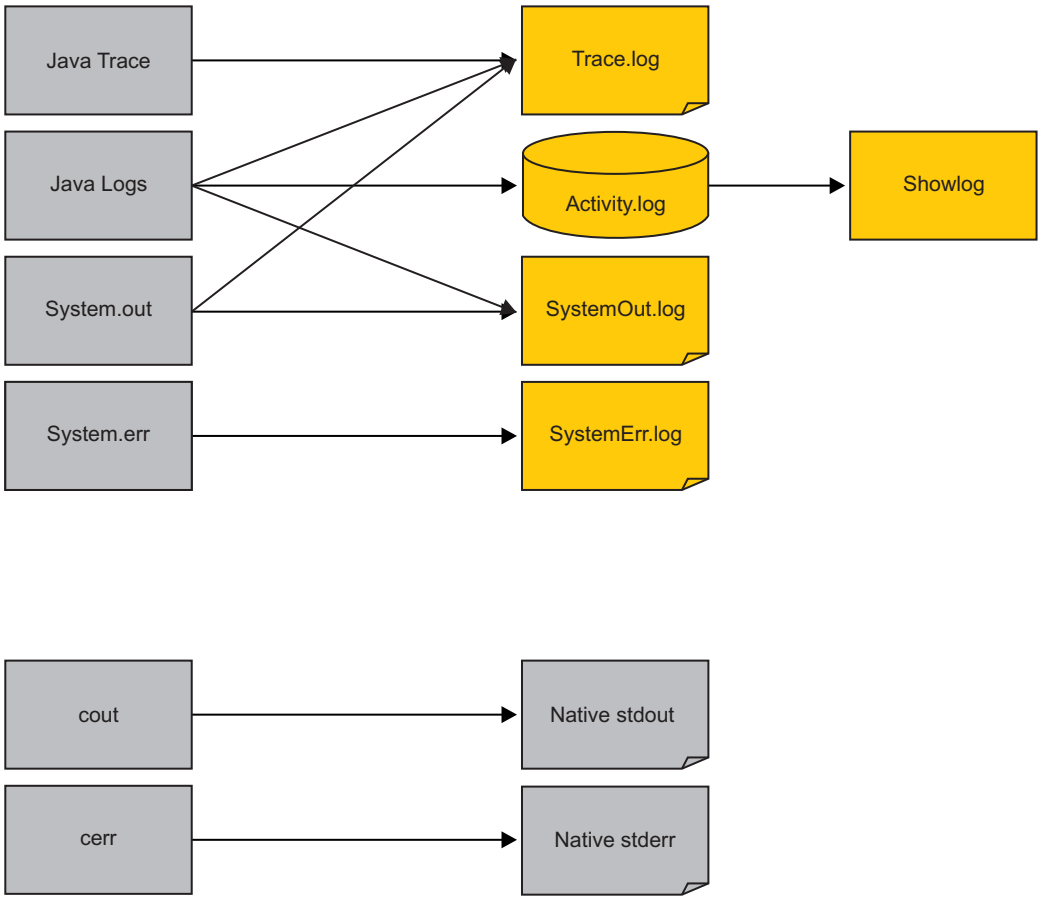


Figure 1. Distributed and IBM i - Basic log and trace mode. Distributed and IBM i - Basic log and trace mode

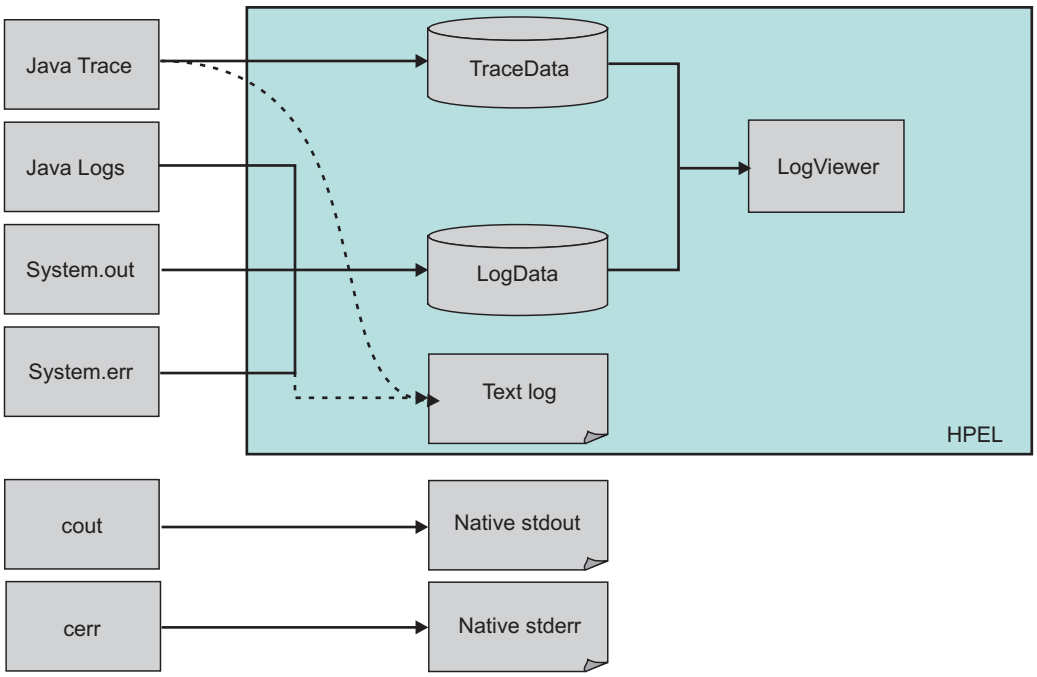


Figure 2. Distributed and IBM i - HPEL log and trace mode. Distributed and IBM i - HPEL log and trace mode

Table 18. Log and trace output for z/OS. This table lists the expected output for logging and tracing output streams when the application server is installed on z/OS.

Log or trace stream	Expected output - traditional log and trace mode	Expected output - HPEL log and trace mode
Java trace	SYSPRINT **Note:** Trace information, which includes events at the Fine, Finer and Finest levels, is written only to this output. If you do not enable diagnostic trace, setting the log detail level to Fine, Finer, or Finest does not affect the logged data.	• HPEL trace repository • HPEL TextLog*.log, when you enable this log
Java logs (Audit level)	Transformed into write-to-operator (WTO) output to the hard copy logging stream	Same outputs as traditional log and trace mode, plus the following: • HPEL log repository • HPEL TextLog*.log, when you enable this log
Java logs (other levels)	SYSOUT or your configured error stream	Same outputs as traditional log and trace mod, plus the following: • HPEL log repository • HPEL TextLog*.log, when you enable this log
System.out	SYSPRINT	• HPEL log repository • HPEL TextLog*.log, when you enable this log
System.err	SYSOUT	• HPEL log repository • HPEL TextLog*.log, when you enable this log
cout (the C or C++ output stream)	SYSPRINT	SYSPRINT
cerr (the C or C++ error stream)	SYSOUT	SYSOUT

Table 18. Log and trace output for z/OS (continued). This table lists the expected output for logging and tracing output streams when the application server is installed on z/OS.

Log or trace stream	Expected output - traditional log and trace mode	Expected output - HPEL log and trace mode
Native trace	SYSPRINT **Note:** Trace information, which includes events at the Fine, Finer and Finest levels, is written only to this output. If you do not enable diagnostic trace, setting the log detail level to Fine, Finer, or Finest does not affect the logged data.	SYSPRINT **Note:** Trace information, which includes events at the Fine, Finer and Finest levels, is written only to this output. If you do not enable diagnostic trace, setting the log detail level to Fine, Finer, or Finest does not affect the logged data.
Native Message logs	SYSOUT or your configured error stream	SYSOUT or your configured error stream

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

When considering the direction of log and trace streams, keep in mind the following acronyms and abbreviations:

cerr C or C++ error steam

cout C or C++ output steam

DD Data Description statements

HFS Hierarchical File System

JCL Job Control Language

JES Job Entry Subsystem

MVS Multiple Virtual Storage

WTO Write-to-operator

WTOR
Write-to-operator with reply

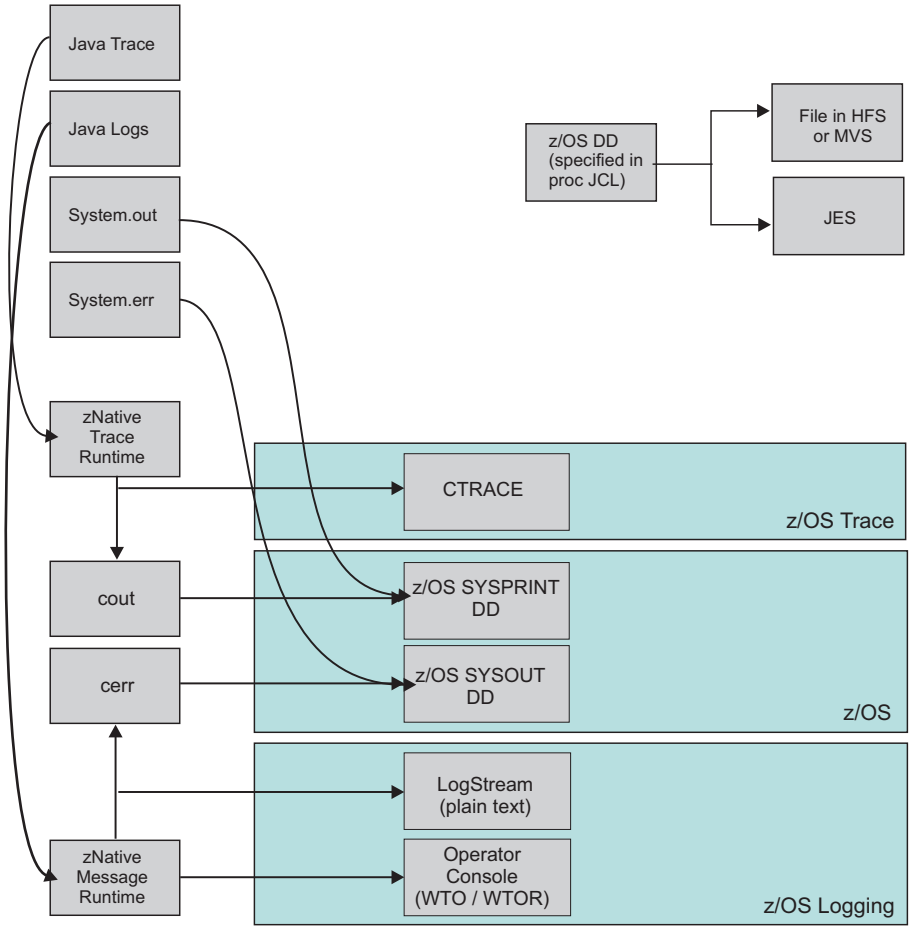


Figure 3. z/OS - Traditional log and trace mode. z/OS - Traditional log and trace mode

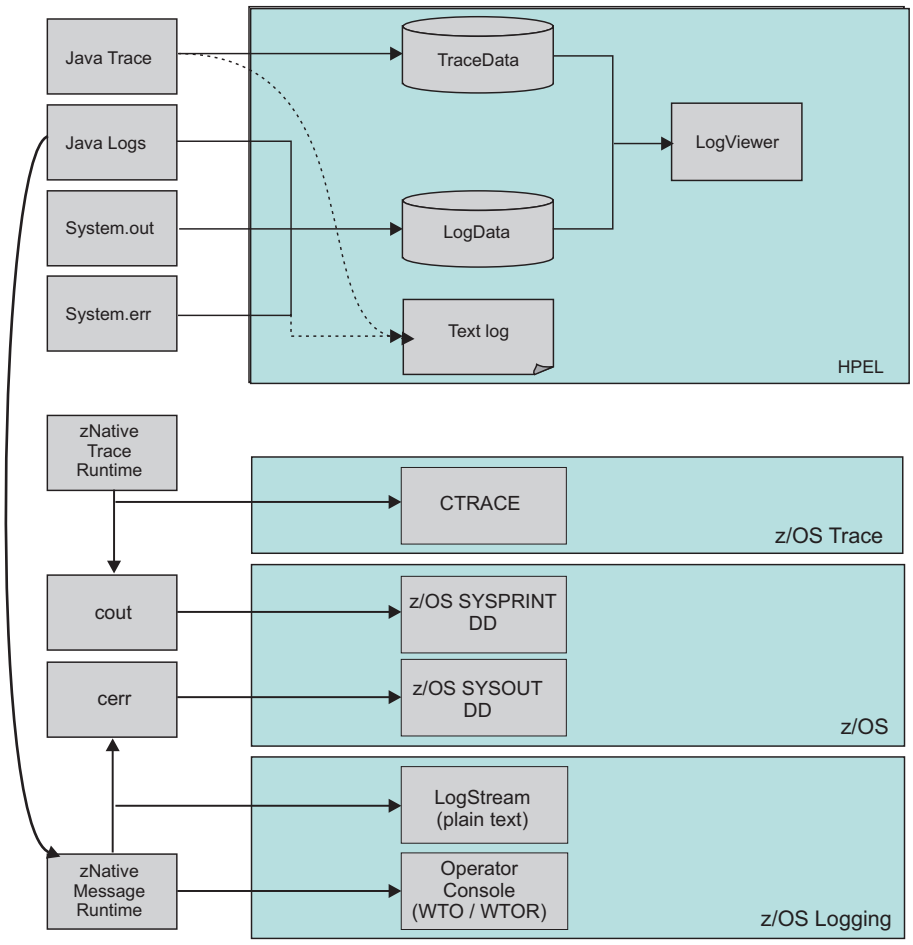


Figure 4. z/OS - HPEL log and trace mode. z/OS - HPEL log and trace mode

Log level settings

Use this topic to configure and manage log level settings.

To view this administrative console page, click **Troubleshooting** > **Logs and trace** > ***server_name*** > **Change log detail levels**

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.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Disable logging and tracing of potentially sensitive data

The application server has a list of loggers which are known to potentially write sensitive information to the log and trace when enabled. For example, enabling certain HTTP related loggers at FINEST level may result in confidential user-specified information from HTTP requests being stored in the trace files. If you want the server to avoid enabling these loggers at levels which are known to be used for potentially sensitive information, check the **Disable logging and**

tracing of potentially sensitive data check box. When the server is started, or when the log detail level specification is modified at run time, the server will compare the list of loggers and levels specified in the log detail level specification to the list of loggers and levels in the sensitive logger list, and will update the log detail level specification as needed.

Change Log Detail Levels

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

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

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

The format of the log detail level specification is:

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

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

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

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

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

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

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

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

Note: Logging level values are case-sensitive and begin with a lowercase letter.

Table 19. Valid logging levels. The following table lists the valid levels for application servers at WebSphere Application Server Version 6 and later.

Version 6 and later logging level	Content / Significance
off	Logging is turned off.
fatal	Task cannot continue and component, application, and server cannot function.
severe	Task cannot continue but component, application, and server can still function. This level can also indicate an impending unrecoverable error.
warning	Potential error or impending error. This level can also indicate a progressive failure (for example, the potential leaking of resources).

Table 19. Valid logging levels (continued). The following table lists the valid levels for application servers at WebSphere Application Server Version 6 and later.

Version 6 and later logging level	Content / Significance
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, and return values
finer	Trace information - Detailed trace
finest	Trace information - A more detailed trace that includes all the detail that is needed to debug problems
all	All events are logged. If you create custom levels, All includes those levels, and can provide a more detailed trace than finest.

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

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

Correlation

Specify correlation settings that you would like to enable. Select the **Enable log and trace correlation** check box to enable correlation for the application server. Clear the **Enable log and trace correlation** check box to disable correlation for the application server. Select **Include request IDs in log and trace records**, **Include request IDs in log and trace records and correlation log records**, or **Include request IDs in log and trace records, create correlation log records, and capture data snapshots**, as appropriate.

Note: Enable XCT to include request IDs in log and trace files when you want to see which log and trace entries, in all threads and application server processes, are related to the same request. Request IDs are only recorded when using HPEL log and trace mode and can be seen or used for filtering using the logViewer command.

Note: Enable XCT to create correlation log records when you want to log how requests branch between threads and processes, and see extra information about each request. Enabling XCT to create correlation log records might have a significant performance impact on your system, so is best suited to test and development environments.

Note: Enable XCT to capture data snapshots when you want to store entire request and response bodies to the file system. Enabling XCT to capture data snapshots might have a significant performance impact on your system, so is best suited to test and development environments. XCT captures data snapshots for message requests and responses handled by the SIBus.

Note: Data snapshots are captured and written to the $*SERVER_LOG_ROOT*/snapdata directory. The application server does not automatically clean up files from this directory. You will need to delete the files from this directory periodically when data snapshot capturing is enabled. Data snapshots store entire request and response contents and may include sensitive information. This option might not be appropriate for use in production environments.

Save runtime changes to configuration as well

Specifies that changes are made to both the dynamic state of the running server, and the server configuration, which takes effect on the next restart. If this check box is not selected, the server does not copy the settings into the server configuration.

Changing the message IDs used in log files

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

Before you begin

Note: Beginning with WebSphere Application Server Version 6.0, logging files are formatted according to a standardized system. However, the default runtime behavior is still configured to use the older format. In new releases of WebSphere Application Server, the message IDs that are written to log files will be changed to ensure they do not conflict with other IBM products. The default runtime behavior is still configured to use the older message IDs, deprecated in Version 8.5.

As a result of the default runtime behavior, you might see a mixture of messages that use 4–letter message prefixes and 5–letter message prefixes. The information in this topic explains how to change your configuration so that the messages consistently show with 5–letter message prefixes. The default behavior has not changed to minimize the impact on customers that depend on the existence of the 4–letter message prefixes.

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

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

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

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

About this task

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

Procedure

- To configure logging files so that they use the newer, 5–letter error message prefixes for each process, use the following commands with the wsadmin utility:
  - Using Jacl:

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

  - Using Jython:

```
ls = java.lang.System.getProperty("line.separator")
cfgJvmList = AdminConfig.list("JavaVirtualMachine").split(ls)
print cfgJvmList
cfgJvm = cfgJvmList[JavaVirtualMachine]
AdminConfig.create('Property', cfgJvm, [['name', 'com.ibm.websphere.logging.messageId.version'], ['value', '6'],
['required', 'false']])
AdminConfig.save()
```

    Where *JavaVirtualMachine* is the number of the process that you want to use.

    When you specify the process, the first process listed is zero (0), the second process is one (1), and so on. Make the changes for each JVM in the cell for consistent output formatting.

Important: Restart the application server for the changes to take effect.

- To change the configuration so that the log files contain the newer, 5–letter message prefixes in the `startServer.log` or `stopServer.log` files, modify the `startServer` and `stopServer` scripts in the *install_root*`/bin` directory.

  Within these scripts, append the following code to the end of the existing D_ARGS parameter:

  ```
  $DEBUG -Dcom.ibm.websphere.logging.messageId.version=6
  ```

Results

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

Converting log files to use IBM unique Message IDs

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

Before you begin

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

About this task

To facilitate the migration of logging tools that are reliant on the old message IDs, the convertlog command is provided to convert the message IDs of log entries from the old standard to the new standard, or the new standard back to the old. By default, the software is configured to use the old message IDs when logging, but you can change the default output with the com.ibm.websphere.logging.messageId.version system property. Read “Changing the message IDs used in log files” on page 99 for more information.

Procedure

Use the convertlog command to convert the log output:

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

Results

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

convertlog command

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

Previous versions of WebSphere Application Server used message IDs that are deprecated in WebSphere Application Server Version 8.5. To facilitate the migration of tools based on the old message IDs, the convertlog command is implemented to translate log files from one message ID standard to the other.

Use the convertlog command as follows:

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

MessageConverter class

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

Use the MessageConverter class with log analysis tools to convert message IDs from earlier versions of WebSphere Application Server into the corresponding message IDs that are used in later releases, or to revert message IDs to an earlier format.

Method:

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

Parameters:

Use the following parameters with the `MessageConverter` class:

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

HTTP error, FRCA, and NCSA access log settings

Use this page to configure the global HTTP error log, and National Center for Supercomputing Applications (NCSA) access log settings for an HTTP inbound channel. If you are running the product on z/OS, you can also use this page to configure the global Fast Response Cache Accelerator (FRCA) log settings for an HTTP inbound channel. A FRCA log is a specialized form of a NCSA log and can only be created in a z/OS environment.

To view this administrative console page, click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***. Under **Troubleshooting**, click **NCSA access and HTTP error logging**. This console page has separate sections for each type of logging. The FRCA logging section only appears if you are running the product on z/OS.

The HTTP error log contains a record of HTTP processing errors that occur. The level of error logging that occurs is dependent on the value that is 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 that are contained in a NCSA access log are in NCSA format.

The FRCA log is a specialized NCSA access log that can only be created if you are running the product on z/OS. This log contains a record of all inbound client requests that are handled by the Fast Response Cache Accelerator. All of the messages that are contained in this log are in NCSA format.

Configuring and enabling the logging is a two step process. After you use this page to configure the logging, you must explicitly enable each type of logging for the appropriate HTTP channels. To view the

settings page for an HTTP channels, click **Servers > Server Types > WebSphere application servers >** *server* **> Web Container Settings > Web container transport chains > Chain > HTTP inbound channel**.

In a z/OS environment, HTTP error, and NCSA access, and FRCA logging must be configured at the controller level.

gotcha: The settings for any of these logs can also be modified on the settings page for a specific HTTP inbound channel. Any changes that you make on the HTTP inbound channel settings page only apply to that specific inbound channel and override any global configuration settings that you specify on this page.

Enable logging service at server start-up

Select this option if you want any of the following logging to start when the server starts:
- FRCA logging
- NCSA access logging
- HTTP error logging

FRCA access logging

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

This field only displays if you are running the product on z/OS.

Enable access logging

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

This field only displays if you are running the product on z/OS.

FRCA log file path

Specifies the directory path and name of the FRCA log. You should use a server-specific variable, such as $(SERVER_LOG_ROOT), to prevent log file name collisions.

This field only displays if you are running the product on z/OS.

FRCA log maximum size

Specifies the maximum size, in megabytes, of the FRCA access log. When the content of the FRCA access log reaches the specified maximum size limit, a <*logname*>.<*timestamp*>.log archive file is created. The current content of the FRCA access log is then copied to this archive log.

An example of a file name for this archive log follows::

```
frca_access_11_09_20_16.15.04.log
```

The next time the content in the FRCA access log reaches the specified maximum log size, the content of the FRCA access log is again copied to the <*logname*>.<*timestamp*>.log archive file. The copy process overwrites the current content of the archive file with the most current content of the FRCA access log.
NOTE: When there are multiple archive logs, as determined by the setting of the "Maximum number of historical files", the oldest archive log is the one overwritten.

This field only displays if you are running the product on z/OS.

Maximum number of historical files

Specifies the maximum number of historical versions of the FRCA log file that are kept for future reference.

This field only displays if you are running the product on z/OS.

FRCA log format
Specifies which FRCA format is used when logging client access information. If you select Common, the log entries contain the requested resource and a few other pieces of information, but does not contain referral, user agent, and cookie information. If you select Combined, referral, user agent, and cookie information is included.

This field only displays if you are running the product for z/OS.

NCSA access logging

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.

On the z/OS platform, you should use a server-specific variable, such as $(SERVER_LOG_ROOT), to prevent log file name collisions.

Access log maximum size
Specifies the maximum size, in megabytes, of the NCSA access log. When the content of the NCSA access log reaches the specified maximum size limit, a <*logname*>.<*timestamp*>.log archive file is created. The current content of the NCSA access log is then copied to this archive log.

An example of a file name for this archive log follows::

```
ncsa_access_11_09_20_16.15.04.log
```

The next time the content in the NCSA access log reaches the specified maximum log size, the content of the NCSA access log is again copied to the <*logname*>.<*timestamp*>.log archive file. The copy process overwrites the current content of the archive file with the most current content of the NCSA access log. **NOTE:** When there are multiple archive logs, as determined by the setting of the "Maximum number of historical files", the oldest archive log is the one overwritten.

Maximum number of historical files
Specifies the maximum number of historical versions of the NCSA access log file that are kept for future reference.

You can use the EnableBuildBackupList HTTP transport custom property to enable the HTTP channel to scan for the history files in the access and error logs directory, and rolling these files over with any newer log files created. See the topic *HTTP Transport channel custom properties* for a description of how to specify this custom property.

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

Entries in the NCSA access log contain a local time stamp.

You can use the HTTP transport channel custom property **accessLogFormat** to customize the format of the NCSA access log for a specific HTTP transport channel. See the topic *HTTP transport channel custom properties* for a description of how to use this custom property.

Error logging

Enable error logging

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

Log file path

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

On the z/OS platform, you should use a server-specific variable, such as $(SERVER_LOG_ROOT), to prevent log file name collisions.

Error log maximum size

Specifies the maximum size, in megabytes, of the HTTP error log. When the content of the HTTP error log reaches the specified maximum size limit, a <*logname*>.<*timestamp*>.log archive file is created. The current content of the HTTP error log is then copied to this archive log.

An example of a file name for this archive log follows::

```
http_access_11_09_20_16.15.04.log
```

The next time the content in the HTTP error log reaches the specified maximum log size, the content of the HTTP error log is again copied to the <*logname*>.<*timestamp*>.log archive file. The copy process overwrites the current content of the archive file with the most current content of the HTTP error log.
NOTE: When there are multiple archive logs, as determined by the setting of the "Maximum number of historical files", the oldest archive log is the one overwritten.

Maximum number of historical files

Specifies the maximum number of historical versions of the Error log file that are kept for future reference.

You can use the EnableBuildBackupList HTTP transport custom property to enable the HTTP channel to scan for the history files in the access and error logs directory, and rolling these files over with any newer log files created. See the topic *HTTP Transport channel custom properties* for a description of how to specify this custom property.

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

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

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

Debug

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

Chapter 6. Using High Performance Extensible Logging to troubleshoot applications

You can use High Performance Extensible Logging (HPEL) to help diagnose problems in WebSphere Application Server.

About this task

Administrators using WebSphere Application Server need to use log and trace files to determine whether their applications and the server are running correctly.

Logs typically contain information that is of interest to administrators and must be looked at periodically to ensure there are no unexpected errors or warnings.

Trace typically contains information that is useful for debugging application or server problems and can help identify specific problems with individual components.

Procedure

1. Enable HPEL if you have not done so already. Read about changing from basic mode to HPEL logging and tracing for more information.
2. Configure the HPEL facility. For example, configure HPEL to store your logs and trace in appropriate directories, and specify how long you want log and trace content to be retained before being deleted. Read about HPEL to learn about the log and trace framework overall, and read about configuring HPEL for more configuration information.
3. Restart the application server after making static configuration changes.

High Performance Extensible Logging (HPEL)

High Performance Extensible Logging (HPEL) is a log and trace facility that is provided as a part of WebSphere Application Server.

Overview

Note: The basic log and trace facility is enabled by default. To use HPEL you must enable it.

HPEL provides a convenient mechanism for storing and accessing log, trace, System.err, and System.out information produced by the application server or your applications. It is an alternative to the existing log and trace facilities offered on the z/OS platform which exploit JES, LogStreams, Component Trace, Hierarchical File System, or other facilities.

HPEL log and trace storage

HPEL provides a log data repository, a trace data repository, and a text log file. See the following figure to understand how applications and the application server store log and trace information.

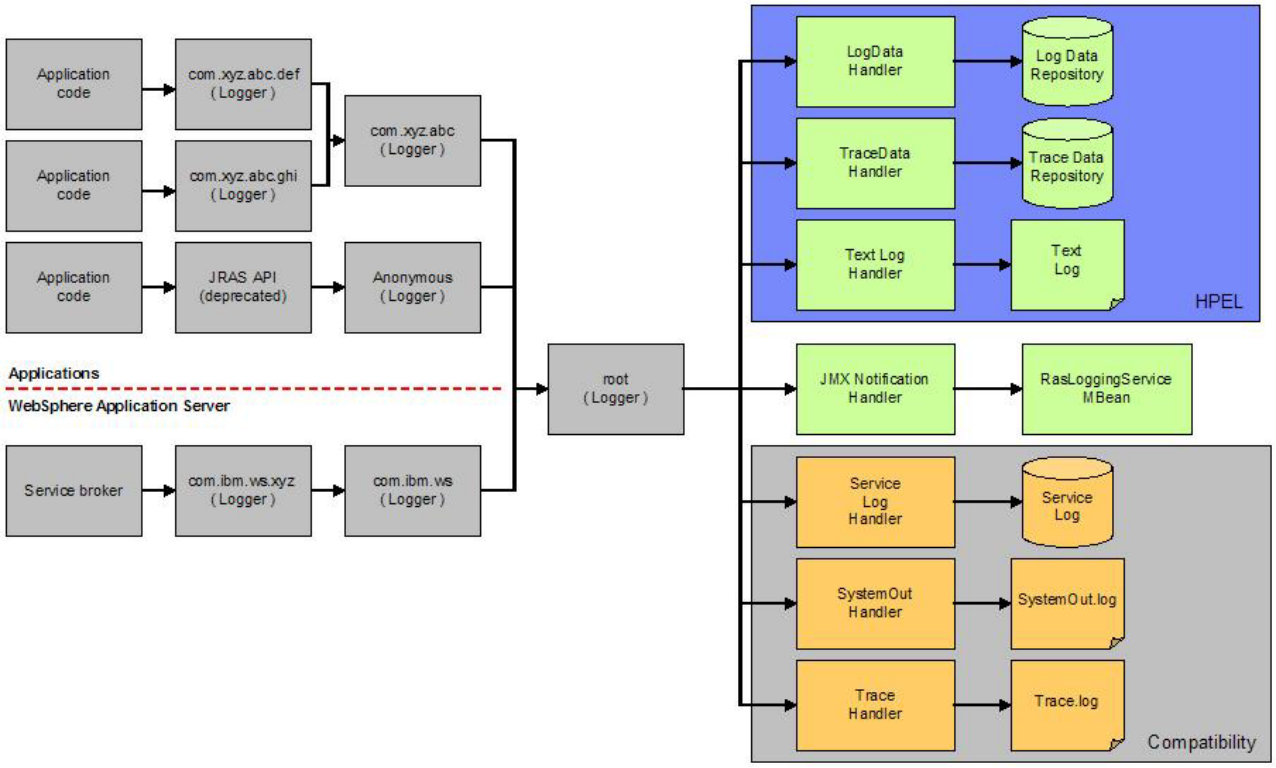


Figure 5. Log and trace storage for HPEL and basic logging.

HPEL log data repository

The log data repository is a storage facility for log records. Log data is typically intended to be reviewed by administrators. This includes any information applications or the server write to System.out, System.err, or java.util.logging at level Detail or higher (including Detail, Config, Info, Audit, Warning, Severe, Fatal, and any custom levels at level Detail or higher).

HPEL trace data repository

The trace data repository is a storage facility for trace records. Trace data is typically intended for use by application programmers or by the WebSphere Application Server support team. This includes any information applications or the server write to java.util.logging at levels below level Detail (including Fine, Finer, Finest, and any custom levels below level Detail).

Note: Log and trace content written to the deprecated JRAS logging API is also included in the log and trace data repositories. Some logging APIs, such as Jakarta Commons Logging can also be configured to route their log and trace data to java.util.logging, and would have their output stored in the log data or trace data repository as well.

HPEL text log

The text log file is a plain text file for log and trace records. The text log file is provided for convenience, primarily so that log content can be read without having to run the LogViewer command-line tool to convert the log data repository content to plain text.

The text log file does not contain any content that is not also stored in either the log data repository or trace data repository. You can disable the text log to enhance server performance. The text log can be configured to record trace content for debugging convenience.

The text log file only contains log entries that are generated by the controller process and not those from servant or adjunct processes. As such, application log records are not written to the

text log on z/OS. To view log or trace data for all application server processes, use the LogViewer command-line tool or the HPEL log viewing tool in the administrative console.

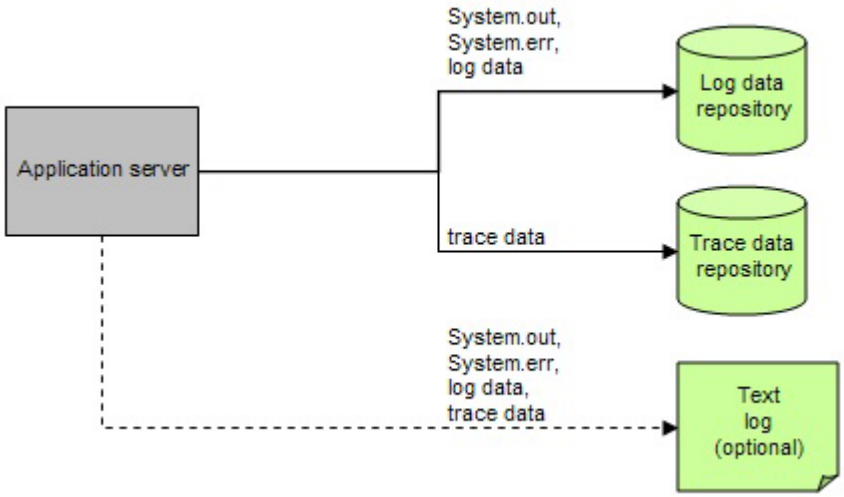
Note: Writing trace to the text log is expensive from a performance perspective.

Log and trace performance

Log and trace events are each stored in only one place

Log events, System.out, and System.err are stored in the log data repository. Trace events are stored in the trace data repository. If the text log file is disabled, HPEL might only write log and trace content to these repositories. Storing each type of event in one place ensures that performance is not wasted on redundant data storage.

Log events, and optionally trace events, are written to the text log file when it is enabled. Since this data is always also stored in the log data and trace data repositories, the text log file content is redundant. The text log is convenient for users who do not want to run the LogViewer command-line tool to see their logs and trace; but you can disable the text log if this convenience is not needed.



Log and trace repositories are not shared across processes

Synchronizing activities between processes causes a degradation in performance to all processes involved. With HPEL, each server process has its own log data repository, trace data repository, and text log file. Since these files are not shared across processes, the server runtime environment does not need to synchronize with other processes when writing to these destinations.

Data is not formatted unless it is needed

Formatting data for an user to read uses processor time. Rather than format log event and trace event data at run time, HPEL log and trace data is stored more rapidly in a proprietary binary representation. This improves the performance of the log and trace facility. By deferring log and trace formatting until the LogViewer is run, sections of the log or trace that are never viewed are never formatted.

You can enable the text log file, which stores the log data and trace data in an already readable text format.

Note: Disable the text log when performance of your server is a key concern, or if the text log is not wanted.

Log and trace data is buffered before being written to disk

Writing large blocks of data to a disk is more efficient than writing the same amount of data in small blocks. HPEL provides buffer log and trace data before writing it to disk. By default, log and trace data is stored in an 8 KB buffer before being written to disk. If the buffer is filled within 10 seconds, the buffer is written to disk. If the buffer is not filled within that time it is automatically written to disk to ensure that the logs have the most current information.

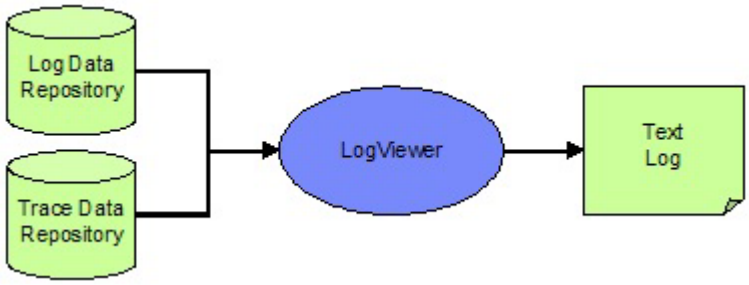
Note: The size of the buffer can be controlled by setting the HPEL.BUFFER.SIZE system property. The frequency that the buffer writes to disk can be controlled by setting the HPEL.FLUSH.PERIOD.SECS system property.

Administration of log and trace

HPEL has been designed to be easy to configure and understand. For example, administrators can easily configure how much disk space to dedicate to logs or trace, or how long to retain log and trace records, and leave the management of log and trace content up to the server. As another example all log, trace, System.out, and System.err content can be accessed using one easy-to-use command (LogViewer), avoiding any possible confusion over which file to access for certain content.

Reading from the log data and trace data repositories

The log data and trace data repositories are stored in a WebSphere Application Server proprietary format and cannot be read using text file editors such as Notepad or VI. You can copy the log data and trace data repositories in to a plain text format using the LogViewer command.



HPEL LogViewer command

The HPEL LogViewer is an easy-to-use, command-line tool provided for HPEL users to work with the log data and trace data repositories. The LogViewer provides filtering and formatting options that make finding important content in the log data and trace data repositories easy. For example, a user might filter any errors or warnings, then filter all log and trace entries that occurred within 10 seconds of a key error message on the same thread.

Filtering using log and trace record extension content

Note: You can use the LogViewer command-line tool to filter records based on the content of log and trace record extensions. The application server automatically creates an appName extension for each log and trace record related to a Java Platform, Enterprise Edition (Java EE) application, indicating the name of that application. The application server also automatically creates a requestID extension for each log and trace record created during the processing of certain types of requests (for example HTTP or JMS requests), indicating the unique ID of that request.

The requestID extension is added only to log and trace records when Cross-Component Trace is enabled. HPEL also provides the ability for developers to add custom extensions to log and trace records using a log record context API (com.ibm.websphere.logging.hpel.LogRecordContext).

HPEL in the administrative console

The administrative console contains pages that enable HPEL administrators to:

- Configure the HPEL log data repository.

- Configure the HPEL trace data repository.
- Configure the HPEL text log file.
- View the contents of the HPEL log and trace data repositories.
- View and set the log detail levels for logging and tracing.

To use these capabilities, in the administrative console, click **Troubleshooting** > **Logs and Trace** link.

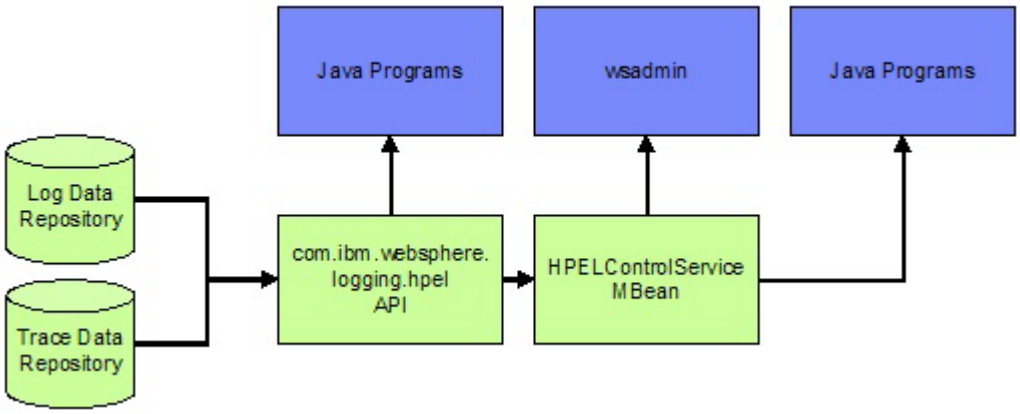
Development resources

HPEL has been designed to make working with log and trace content more flexible and effective than the basic logging facility. Log and trace content can be easily filtered to show only the records that are of interest. You can use the command line (see the description of the HPEL LogViewer command), or developers can create powerful log handling programs using the HPEL API.

Scripts and Java programs read from the log data and trace data repositories

Developers and scripters have a number of options for how to read the log data and trace data repositories:

- Locally or remotely from a **wsadmin** script, using the HPELControlService JMX MBean
- Locally or remotely from a Java program, using the HPELControlService JMX MBean
- Locally from a Java program, using the com.ibm.websphere.logging.hpel API



HPEL-related JMX MBeans

A MBean interface has been provided to make it easy to access HPEL repository content remotely. For example, a developer might write a JMX client program to read log content from across their WebSphere Application Server cell. This interface is part of the HPELControlService MBean. Refer to the MBean interface documentation for details on the HPEL remote log reading interface.

Table 20. JMX MBeans related to HPEL. The JMX MBeans are related to the operation of HPEL.

JMX MBean	Description
HPELControlService	Provides operations related to configuring the log or trace detail level of the server, viewing the log component registry, and querying the log and trace repositories
HPELLogDataService	Provides operations related to configuring the log data repository of the server
HPELTraceDataService	Provides operations related to configuring the trace data repository of the server
HPELTextLogService	Provides operations related to configuring the text log file of the server
RasLoggingService	Only used for JMX Notification of log events

When using HPEL for log and trace rather than basic logging, the log and trace JMX MBean, TraceService, is not used.

HPEL API

An API has been provided to make it easy for developers to develop tools to consume content from the HPEL log and trace repositories. For example, a developer might write a Java program to search the log and trace content to find any messages with message IDs that match a known list of important message IDs. This API is in the com.ibm.websphere.logging.hpel package. Refer to the API documentation for details on the HPEL log reading API.

Log and trace record extensibility

Note: Developers can use HPEL to add custom extensions to log and trace records through a log record context API (com.ibm.websphere.logging.hpel.LogRecordContext). When HPEL stores log and trace records, it includes any extensions present in the log record context on the same thread. For example, a developer might write a servlet filter to add important HTTP request parameters to the log record context. While that servlet runs, HPEL adds those extensions to any log and trace records created on the same thread.

As with other log and trace record fields, developers can access the record extensions using the HPEL API. This is useful when writing tools to read from log and trace repositories. Developers can also make use of the log record context API to access extensions in custom log handlers, filters, and formatters at run time.

Basic mode and HPEL mode

Two modes of logging and tracing exist in the product, which are basic mode and High Performance Extensible Logging (HPEL) mode. Use this topic to understand the differences between these modes.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Basic mode, the default mode, is the existing log and trace framework from prior releases of WebSphere Application Server. Any existing scripts and tools you have that use logs and trace might need minor modifications to work with HPEL mode.

To take advantage of the new log and trace framework, HPEL mode must be enabled. Once HPEL mode is enabled the JVM logs (typically SystemOut.log and SystemErr.log), the trace log (typically trace.log), and the service log (typically activity.log) are no longer written to. Instead, log and trace content is written to a log data or trace data repository in a proprietary binary format and, if configured, to a text log file. Disabling the writing of the text log file results in the largest possible performance benefit of HPEL. A log viewing tool, LogViewer, is provided to allow for viewing, filtering, monitoring, and formatting the log and trace data in the repositories.

The following figure shows the files used by the basic mode and HPEL mode log and trace facilities. When enabled, the HPEL text log file stores content from Java trace (optional), Java logs, System.out, and System.err. You can disable the HPEL text log in cases where it is not needed as indicated by the dotted lines.

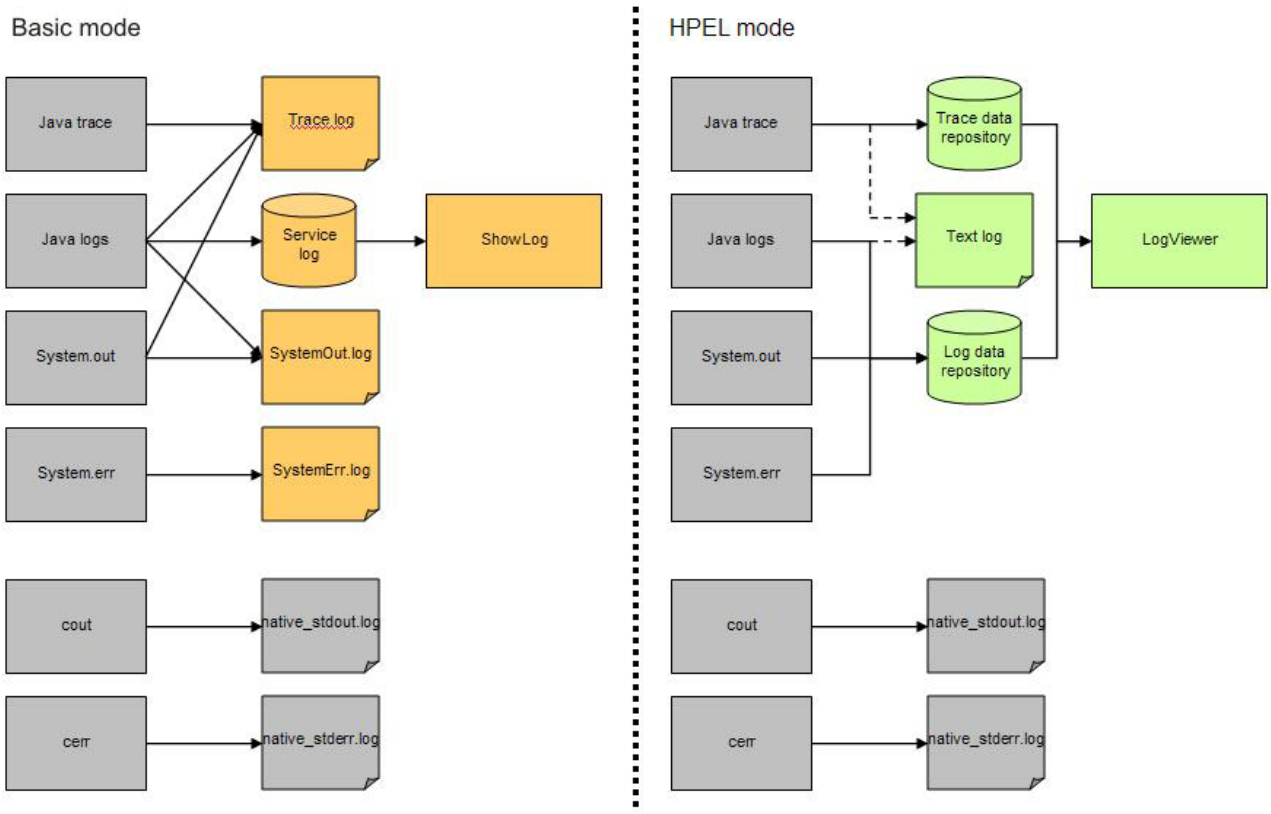


Figure 6. Log and trace files

Table 21. Basic mode and HPEL mode files. This table shows where each of the application and application server log, trace, JVM log, and native JVM log content is routed and how to view that content.

Source	Basic mode files	HPEL mode files	How to view the HPEL files
System.out	SystemOut.log trace.log (when trace enabled)	logdata/*.wbl TextLog_<timestamp>.log (when text log enabled)	logdata - use LogViewer, with optional filtering, to render log data repository as readable text, then use any text editor TextLog - any text editor
System.err	SystemErr.log	Same as System.out	Same as System.out
java.util.logging (levels DETAIL and above)	SystemOut.log activity.log trace.log (when trace enabled)	Same as System.out	Same as System.out
java.util.logging (levels below DETAIL)	trace.log	tracedata/*.wbl TextLog_<timestamp>.log (when text log enabled)	tracedata - use LogViewer, with optional filtering, to render trace data repository as readable text, then use any text editor TextLog - any text editor
native cout	native_stdout.log	native_stdout.log	Any text editor
native cerr	native_stderr.log	native_stderr.log	Any text editor

The following table describes the MBean services:

Table 22. Basic mode and HPEL mode MBeans. This table shows the JMX MBeans that are associated with the basic mode and HPEL mode log and trace frameworks.

Basic mode MBeans	HPEL mode MBeans	HPEL MBean Descriptions
RasLoggingServiceMBean	RasLoggingServiceMBean	Provides JMX Notification
	HPELLogDataServiceMBean	Configures the log repository such as location, retention policy, out of space behavior, buffering, and file switching
	HPELTraceDataServiceMBean	Configures the trace repository such as location, retention policy, out of space behavior, buffering, and file switching
	HPELTextLogServiceMBean	Configures the text log such as location, retention policy, out of space behavior, buffering, file switching, and SystemErr or SystemOut format
TraceServiceMBean	HPELControlServiceMBean	Configures trace specification levels, and provides access to log and trace repository content

Changing from basic mode to HPEL logging and tracing

The basic mode log and trace framework is enabled by default when you set up a new application server. Use this topic to switch to the High Performance Extensible Logging (HPEL) log and trace framework.

Before you begin

Before beginning this task, read about the differences between HPEL mode and basic mode. Be aware of changes you might need to make to any tools and scripts you have that use the basic mode log and trace files.

About this task

HPEL provides faster log and trace handling capabilities and more flexible ways to use log and trace content than the basic mode. You can switch to HPEL mode using the administrative console, or using wsadmin scripting.

Procedure

- Use the administrative console to switch to HPEL.
  1. Log on to the administrative console.
  2. If using an admin agent topology select a node that you want to manage and navigate to it.
  3. From the navigation section in the console choose **Troubleshooting** > **Logs and trace** .
  4. Select the server that you want to switch to HPEL.
  5. Click **Switch to HPEL Mode** .
  6. Save the changes.
- Use wsadmin scripting to switch to HPEL. Complete these steps to modify the server configuration.
  1. Start wsadmin. In this case, you can connect wsadmin to a running server or access the configuration data for a stopped server. Read about starting the wsadmin scripting client for more information.
  2. Get a reference to the HighPerformanceExtensibleLogging configuration object.

     Using Jython:

     ```
     HPELService = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:
     myServer/HighPerformanceExtensibleLogging:/")
     ```

Table 23. AdminConfig command description. The table lists AdminConfig command and description.

Command	Description
myCell	The name of the cell
myNode	The host name of the node
myServer	The name of the server

3. Set the HighPerformanceExtensibleLogging enable attribute to true.
   Using Jython:
   ```
   AdminConfig.modify(HPELService, "[[enable true]]")
   ```
4. Get a reference to the RASLoggingService object.
   Using Jython:
   ```
   RASLogging = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:myServer/RASLoggingService:/")
   ```

Table 24. AdminConfig command description. The table lists AdminConfig command and description.

Command	Description
myCell	The name of the cell
myNode	The host name of the node
myServer	The name of the server

5. Set the RASLoggingService enable attribute to false.
   Using Jython:
   ```
   AdminConfig.modify(RASLogging, "[[enable false]]")
   ```
6. Save the configuration.
   Using Jython:
   ```
   AdminConfig.save()
   ```

Results

The server is now configured to use HPEL when you restart.

If this task was done using the deployment manager, you might need to synchronize the node agent on the target node before restarting the server.

What to do next

Configure HPEL to meet your needs. If you have any tools or scripts that were using the basic mode log and trace files, you might need to modify them to continue working with HPEL.

Changing from HPEL to basic mode logging and tracing

Use this topic if you need to switch from HPEL to the basic mode log and trace framework. The basic mode log and trace framework is enabled by default when you set up a new application server.

Before you begin

Before beginning this task, read about the differences between HPEL mode and basic mode. Be aware of changes you might need to make to any tools and scripts you have that use HPEL files or commands.

About this task

You can switch to basic mode using the administrative console, or using wsadmin scripting. HPEL provides faster log and trace handling capabilities and more flexible ways to exploit log and trace content than the basic mode.

Procedure

- Use the administrative console to switch to basic mode.
  1. Log on to the administrative console.
  2. If using an administrative agent topology, select a node that you want to manage and navigate to it.
  3. From the navigation section in the console click **Troubleshooting > Logs and trace**.
  4. Select the server that you want to switch to basic mode.
  5. Click **Change log and trace mode**.
  6. Click **Switch back to basic mode**.
  7. Save the changes.
- Use wsadmin scripting to switch to basic mode. These steps modify the server configuration. The server does not need to be running to perform these steps.
  1. Start wsadmin. In this case, wsadmin can be connected to a running server or accessing the configuration data for a stopped server. Read about starting the wsadmin scripting client for more information.
  2. Get a reference to the HighPerformanceExtensibleLogging configuration object.

     Using Jython:

     ```
     HPELService = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:
     myServer/HighPerformanceExtensibleLogging:/")
     ```

Table 25. AdminConfig command description. The table lists AdminConfig command and description.

Command	Description
myCell	The name of the cell
myNode	The host name of the node
myServer	The name of the server

  3. Set the HighPerformanceExtensibleLogging enable attribute to false.

     Using Jython:

     ```
     AdminConfig.modify(HPELService, "[[enable false]]")
     ```
  4. Get a reference to the RASLoggingService object.

     Using Jython:

     ```
     RASLogging = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:
     myServer/RASLoggingService:/")
     ```

Table 26. AdminConfig command description. The table lists AdminConfig command and description.

Command	Description
myCell	The name of the cell
myNode	The host name of the node
myServer	The name of the server

  5. Set the RASLoggingService enable attribute to true.

     Using Jython:

     ```
     AdminConfig.modify(RASLogging, "[[enable true]]")
     ```
  6. Save the configuration.

Using Jython:

```
AdminConfig.save()
```

Results

The server is now configured to use the basic mode log and trace framework when you restart.

If this task was done using the deployment manager, you might need to synchronize the node agent on the target node before restarting the server.

Determining which of basic mode and HPEL mode is enabled

WebSphere Application Server offers both a High Performance Extensible Logging (HPEL) log and trace framework, and a basic log and trace framework. There are a number of ways to determine which of the two frameworks is enabled.

About this task

In many circumstances it might be important to know whether the HPEL or basic log and trace framework is enabled, for example, when writing a script whose purpose is to read from any log files of the server.

Since the configuration files of a server can differ from the running state of a server (for example when configuration changes have been made but the server has not been restarted), steps are provided for determining the log and trace mode in various ways.

Procedure

- Use wsadmin to determine the log and trace mode that a running server uses.
  1. Start wsadmin. In this case, wsadmin must be connected to a running server, for example through the SOAP port. Read about starting the wsadmin scripting client for more information.
  2. Determine whether the HPELControlService object is available. If the HPELControlService is present it can be concluded that the server is running with the HPEL log and trace framework.

     Using Jython:

     ```
     HPELMBean = AdminControl.queryNames('cell=myCell,node=myNode,
     type=HPELControlService,process=myServer,*')
     if (HPELMBean == ''):
         print "HPEL is not enabled"
     else:
         print "HPEL is enabled"
     ```

Table 27. AdminControl command description. The table lists AdminControl command and their description.

Command	Description
myNode	The host name of the node
myServer	The name of the server

- Use wsadmin to determine the log and trace mode that a servers configuration specifies.
  1. Start wsadmin. In this case, wsadmin can be connected to a running server or accessing the configuration data for a stopped server. Read about starting the wsadmin scripting client for more information.
  2. Determine whether the RASLoggingService configuration object is enabled. If the RASLoggingService config object is enabled then it can be concluded that the server is configured to run with the basic log and trace framework. Otherwise, if the HighPerformanceExtensibleLogging config object is enabled it can be concluded that the server is configured to run with the HPEL log and trace framework.

Note: If both the RASLoggingService config object and the HighPerformanceExtensibleLogging config object are enabled it can be concluded that the server is configured to run with the basic log and trace framework.

Using Jython:

```
RASLogging = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:myServer/RASLoggingService:/")
basicEnabled = AdminConfig.showAttribute(RASLogging, "enable")
if (basicEnabled == "true"):
    print "Basic mode logging in effect"
else:
    HPELSvc = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:myServer/HighPerformanceExtensibleLogging:/")
    HpelEnabled = AdminConfig.showAttribute(HPELSvc, "enable")
    if (HpelEnabled == "true"):
        print "HPEL is enabled"
    else:
        print "No logging is enabled"
```

Table 28. AdminControl command description. The table lists AdminControl command and their description.

Command	Description
myCell	The name of the cell
myNode	The host name of the node
myServer	The name of the server

- Use the administrative console to determine the log and trace mode that a running server uses.
  1. Log into the administrative console.
  2. Click **Troubleshooting > Logs and Trace > *myServer*** (where *myServer* is the name of the server you are interested in)
  3. Find the **Switch to HPEL Mode** button. If this button is available, the server is using the basic log and trace framework. Otherwise, the server is using HPEL.

Results

For any method selected, the result is that you now know whether a server is configured to use the HPEL or basic mode log and trace framework.

Configuring HPEL with wsadmin scripting

You can configure the High Performance Extensible Logging (HPEL) log and trace framework using wsadmin scripting. Use the examples in this topic as a guide to build your own wsadmin scripts.

About this task

HPEL provides faster log and trace handling capabilities and more flexible ways to use log and trace content than the basic mode. You can configure the HPEL mode using the administrative console, or using wsadmin scripting. The examples in this topic show how to configure HPEL using wsadmin. If you complete this task using the deployment manager, then you might need to synchronize the node agent on the target node and restart the server before configuration changes take effect.

Table 29. Variable Names. The table applies to all examples in this topic. All examples use the Jython scripting language.

Variable	Description
myCell	The name of the cell
myNode	The host name of the node
myServer	The name of the server

Procedure

- Use the AdminConfig object to configure HPEL.

  Changes you make using the AdminConfig object take effect the next time you start the server.

  1. Change the trace specification.

     The following example shows how to change the trace specification to *=info:com.ibm.ws.classloader.*=all

     ```
     HPELService = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:myServer/
     HighPerformanceExtensibleLogging:/")
     AdminConfig.modify(HPELService, "[[startupTraceSpec *=info:com.ibm.ws.classloader.*=all]]")
     AdminConfig.save()
     ```

  2. Change the size of the log repository.

     The following example shows how to set HPEL to automatically delete the oldest log content from the log repository when the repository size approaches 65 MB. Specify HPELTrace or HPELTextLog instead of HPELLog to change the setting for the HPEL trace repository or HPEL text log.

     ```
     HPELService = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:myServer/
     HighPerformanceExtensibleLogging:/")
     HPELLog = AdminConfig.list("HPELLog", HPELService)
     AdminConfig.modify(HPELLog, "[[purgeMaxSize 65]]")
     AdminConfig.save()
     ```

  3. Change the log repository location.

     The following example shows how to change the HPEL log repository directory name to /tmp/myDirectory. Specify HPELTrace or HPELTextLog instead of HPELLog to change the setting for the HPEL trace repository or HPEL text log.

     ```
     HPELService = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:myServer/
     HighPerformanceExtensibleLogging:/")
     HPELLog = AdminConfig.list("HPELLog", HPELService)
     AdminConfig.modify(HPELLog, "[[dataDirectory  /tmp/myDirectory]]")
     AdminConfig.save()
     ```

  4. Disable log record buffering.

     The following example shows how to change the HPEL log repository to not use log record buffering. Specify HPELTrace or HPELTextLog instead of HPELLog to change the setting for the HPEL trace repository or HPEL text log.

     ```
     HPELService = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:myServer/
     HighPerformanceExtensibleLogging:/")
     HPELLog = AdminConfig.list("HPELLog", HPELService)
     AdminConfig.modify(HPELLog, "[[bufferingEnabled false]]")
     AdminConfig.save()
     ```

     Note: Enable log record buffering in almost all cases. Only disable log record buffering when your server is failing unexpectedly and cannot write buffered content to disk before stopping.

  5. Start writing to a new log file each day at a specified time.

     The following example shows how to enable the HPEL log repository to start a new log file each day at 3pm. Specify HPELTrace or HPELTextLog instead of HPELLog to change the setting for the HPEL trace repository or HPEL text log.

     ```
     HPELService = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:myServer/
     HighPerformanceExtensibleLogging:/")
     HPELLog = AdminConfig.list("HPELLog", HPELService)
     AdminConfig.modify(HPELLog, "[[fileSwitchTime 15]]")
     AdminConfig.modify(HPELLog, "[[fileSwitchEnabled true]]")
     AdminConfig.save()
     ```

  6. Change the out of space action for the log repository.

     The following example shows how to change the out of space action for the HPEL log repository. Specify HPELTrace or HPELTextLog instead of HPELLog to change the setting for the HPEL trace repository or HPEL text log.

```
HPELService = AdminConfig.getid("/Cell:myCell/Node:myNode/Server:myServer/
HighPerformanceExtensibleLogging:/")
HPELLog = AdminConfig.list("HPELLog", HPELService)
AdminConfig.modify(HPELLog, "[[outOfSpaceAction PurgeOld]]")
AdminConfig.save()
```

- Use the AdminControl object to configure HPEL. Changes you make using the AdminControl object take effect immediately.
  1. Change the trace specification.

     The following example shows how to change the trace specification to *=info:com.ibm.ws.classloader.*=all

     ```
     HPELControlMBean = AdminControl.queryNames('cell=myCell,node=myNode,
     type=HPELControlService,process=myServer,*')
     AdminControl.setAttribute(HPELControlMBean, "traceSpecification",
     "*=info:com.ibm.ws.classloader.*=all")
     ```

  2. Change the size of the log repository.

     The following example shows how to set HPEL to automatically delete the oldest log content from the log repository when the repository size approaches 65 MB. Specify HPELTraceDataService or HPELTextLogService instead of HPELLogDataService to change the setting for the HPEL trace repository or HPEL text log.

     ```
     HPELLogDataMBean = AdminControl.queryNames('cell=myCell,
     node=myNode,type=HPELLogDataService,process=myServer,*')
     AdminControl.setAttribute(HPELLogDataMBean, "purgeMaxSize", "65")
     ```

  3. Change the log repository location.

     The following example shows how to change the HPEL log repository directory name to /tmp/myDirectory. Specify HPELTraceDataService or HPELTextLogService instead of HPELLogDataService to change the setting for the HPEL trace repository or HPEL text log.

     ```
     HPELLogDataMBean = AdminControl.queryNames('cell=myCell,
     node=myNode,type=HPELLogDataService,process=myServer,*')
     AdminControl.setAttribute(HPELLogDataMBean, "dataDirectory", "/tmp/myDirectory")
     ```

  4. Disable log record buffering.

     The following example shows how to change the HPEL log repository to not use log record buffering. Specify HPELTraceDataService or HPELTextLogService instead of HPELLogDataService to change the setting for the HPEL trace repository or HPEL text log.

     ```
     HPELLogDataMBean = AdminControl.queryNames('cell=myCell,node=myNode,
     type=HPELLogDataService,process=myServer,*')
     AdminControl.setAttribute(HPELLogDataMBean, "bufferingEnabled", "false")
     ```

     Note: Enable log record buffering in almost all cases. Only disable log record buffering when your server is failing unexpectedly and cannot write buffered content to disk before stopping.

  5. Start writing to a new log file each day at a specified time.

     The following example shows how to enable the HPEL log repository to start a new log file each day at 3pm. Specify HPELTrace or HPELTextLog instead of HPELLog to change the setting for the HPEL trace repository or HPEL text log.

     ```
     HPELLogDataMBean = AdminControl.queryNames('cell=myCell,node=myNode,
     type=HPELLogDataService,process=myServer,*')
     AdminControl.setAttribute(HPELLogDataMBean, "fileSwitchTime", "15")
     AdminControl.setAttribute(HPELLogDataMBean, "fileSwitchEnabled", "true")
     ```

  6. Change the out of space action for the log repository.

     The following example shows how to change the out of space action for the HPEL log repository. Specify HPELTraceDataService or HPELTextLogService instead of HPELLogDataService to change the setting for the HPEL trace repository or HPEL text log.

     ```
     HPELLogDataMBean = AdminControl.queryNames('cell=myCell,node=myNode,
     type=HPELLogDataService,process=myServer,*')
     AdminControl.setAttribute(HPELLogDataMBean, "outOfSpaceAction", "PurgeOld")
     ```

Results

HPEL is now configured. If you made changes with the AdminConfig command, restart the server to make the changes take effect.

Configuring HPEL

HPEL logging and trace settings

Use this page to view and configure High Performance Extensible Logging (HPEL) logging and trace settings for the server.

Note: You can only access this page when the server is configured to use HPEL log and trace mode.

To view this administrative console page, click **Troubleshooting** > **Logs and trace** > ***server_name***.

Configure HPEL logging

Use this link to configure HPEL log options. If this server is in running state, then important log configuration values are displayed following the link. These values summarize the current runtime values being used by the server.

Configure HPEL trace

Use this link to configure HPEL trace options. If this server is in running state, then important trace configuration values are displayed following the link. These values summarize the current runtime values being used by the server.

Configure HPEL text log

Use this link to configure HPEL text log options. If this server is in running state, then important text log configuration values are displayed following the link. These values summarize the current runtime values being used by the server.

View HPEL logs and trace

Use this page to view log data from the HPEL repository. You can also use this page to filter and search the repository. You can export the customized view or full repository into a text file or into a new HPEL repository.

Change log detail levels

Use this page to enter a log detail level that specifies the components, packages, or groups to trace, and to configure log and trace correlation settings.

Change log and trace mode

Use this link to switch back to basic mode logging, instead of HPEL mode logging currently enabled for this server.

Manage process logs

WebSphere Application Server processes contain two output streams that are accessible to native code, which runs in a particular process. These streams are the **stdout** and **stderr** streams. Native code, including Java virtual machines (JVM), might write data to these process streams. In addition, you can also configure JVM-provided System.out and System.err streams to write their data to these streams.

NCSA access and HTTP error logging

The NCSA access and HTTP error logging page enables you to configure the log settings for your HTTP server.

HPEL log configuration settings

Use this page to configure High Performance Extensible Logging (HPEL) log settings.

Note: You can only access this page when the server is configured to use HPEL log and trace mode.

To view this administrative console page, click **Troubleshooting > Logs and trace > *server_name* > Configure HPEL logging** .

Directory path

Specifies the directory to which log files are written. A subdirectory, logdata, is created in this directory, and the log files are written to this location.

Note: This path cannot be shared between servers. The server creates a file with a `.owner` extension to help detect when two or more servers happen to be trying to use the same path for HPEL output.

Enable log record buffering

Specifies that the logging system avoids writing to disk each time a log record is created. The logging system creates a buffer that can hold a number of log records, and writes the buffered events when the buffer is full. The logging system also writes the buffered events after a few seconds have passed, even if the buffer is not full.

Selecting this setting significantly improves logging performance; however, if the server stops unexpectedly, the contents might not be written to the log repository.

Note: Enable log record buffering in almost all cases. Only disable log record buffering when your server is failing unexpectedly and cannot write buffered content to disk before stopping.

Start new log file daily at <time>

Enables the logging framework to close the log file and start a new file at the specified time of day. Closing the file makes it easy to copy the file to an archive.

Note: If you want to automatically archive your log files, set up your backup program to copy files after the time you configured for new logs to be started. Configure the backup to occur at least 10 minutes after the time configured for new logs to be started to ensure that the server has closed the previous file.

Begin cleanup of oldest records

Specifies the log cleanup settings to be used to automatically purge the oldest log records, or log records that no longer fit in the configured space, from the log repository.

Select **When log size approaches maximum** to configure automatic log file cleanup to begin when the total size of the log repository approaches the configured maximum size.

Select **When oldest records reach age limit** to configure automatic log file cleanup to begin when log content is the age limit specified.

Select **When either age or size restriction is met** to configure automatic log file cleanup to begin when either of the previous conditions is met.

Regardless of the selection chosen, records are deleted from the log repository in the order in which they were written to the log repository.

Log record age limit:

Specifies the lifespan, in hours, that log records can remain in the log repository before the log records can be automatically deleted by the server. When the oldest records in the log repository have existed longer than the age limit specified, then those records are targeted for deletion by the server.

Maximum log size:

Specifies the maximum total size, in megabytes, that the server allows the log repository to reach. When the log repository approaches this size limit, the server deletes the oldest records from the log repository to make space for new log records.

Out of space action
Specifies how the server reacts to an inability to add content to the log repository.

Select **Stop server** to specify that the server stops when the server is unable to write to the log repository.

Select **Purge old records** to specify that the server continues to run, and that the oldest log records are immediately removed when the server is unable to write to the log repository.

Select **Stop logging** to specify that the server continues to run, but that the server cannot continue to write to the log when the server is unable to write to the log repository.

Save runtime changes to configuration as well
Specifies that changes are made to both the dynamic state of the running server, and the server configuration, which takes effect on the next restart. If this check box is not selected, the server does not copy the settings into the server configuration.

HPEL trace configuration settings
Use this page to configure High Performance Extensible Logging (HPEL) trace settings.

Note: You can only access this page when the server is configured to use HPEL log and trace mode.

To view this administrative console page, click **Troubleshooting > Logs and trace > *server_name* > Configure HPEL trace**.

Trace to a directory
Specifies that the tracing system writes trace records to the trace directory as they are created by the server.

Enable log record buffering
Specifies that the tracing system avoids writing to disk each time a trace record is created. The tracing system creates a buffer that can hold a number of trace records, and writes the buffered events when the buffer is full. The tracing system also writes the buffered events after a few seconds have passed, even if the buffer is not full.

Selecting this setting significantly improves tracing performance; however, if the server stops unexpectedly, the contents might not be written to the trace repository.

Note: Enable trace record buffering in almost all cases. Only disable trace record buffering when your server is failing unexpectedly and cannot write buffered content to disk before stopping.

Start new log file daily at <time>
Enables the tracing framework to close the trace file and start a new file at the specified time of day. Closing the file makes it easy to copy the file to an archive.

Note: If you want to automatically archive your trace files, set up your backup program to copy files after the time you configured for new files to be started. Configure the backup to occur at least 10 minutes after the time configured for new files to be started to ensure that the server has closed the previous file.

Begin cleanup of oldest records

Specifies the trace cleanup settings to be used to automatically purge the oldest trace records, or trace records that no longer fit in the configured space, from the trace repository.

Select **When log size approaches maximum** to configure automatic trace file cleanup to begin when the total size of the trace repository approaches the configured maximum size.

Select **When oldest records reach age limit** to configure automatic trace file cleanup to begin when trace content is the age limit specified.

Select **When either age or size restriction is met** to configure automatic trace file cleanup to begin when either of the previous conditions is met.

Regardless of the selection chosen, records are deleted from the trace repository in the order in which they were written to the trace repository.

Log record age limit:

Specifies the lifespan, in hours, that trace records can remain in the trace repository before the trace records can be automatically deleted by the server. When the oldest records in the trace repository have existed longer than the age limit specified, then those records are targeted for deletion by the server.

Maximum log size:

Specifies the maximum total size, in megabytes, that the server allows the trace repository to reach. When the trace repository approaches this size limit, the server deletes the oldest records from the trace repository to make space for new trace records.

Out of space action

Specifies how the server reacts to an inability to add content to the trace repository.

Select **Stop server** to specify that the server stops when the server is unable to write to the trace repository.

Select **Purge old records** to specify that the server continues to run, and that the oldest trace records are immediately removed when the server is unable to write to the trace repository.

Select **Stop logging** to specify that the server continues to run, but that the server cannot continue to write to the trace when the server is unable to write to the trace repository.

Trace to a memory buffer

Specifies that the tracing system writes trace records to a memory buffer.

You can write the memory buffer contents to the trace directory from the runtime tab.

Memory buffer size

Specifies the amount of memory the tracing system allocates in the server to contain trace records.

In cases where the memory buffer is full when a new trace record is created, the oldest entry from the memory buffer is deleted to make space.

Dump button
Use this button to write the contents of the trace memory buffer to the trace directory.

The tracing system resets the memory buffer after you dump it. The tracing system continues to record trace records in the memory buffer after you dump the buffer.

Directory to use for tracing and dumping memory buffer
Specifies the directory to which trace files are written. A subdirectory, trace data, is created in this directory, and the trace files are written to this location.

Note: This path cannot be shared between servers. The server creates a file with a `.owner` extension to help detect when two or more servers happen to be trying to use the same path for HPEL output.

Save runtime changes to configuration as well
Specifies that changes are made to both the dynamic state of the running server, and the server configuration, which takes effect on the next restart. If this check box is not selected, the server does not copy the settings into the server configuration.

HPEL text log configuration settings
Use this page to configure High Performance Extensible Logging (HPEL) settings for text log.

Note: You can only access this page when the server is configured to use HPEL log and trace mode.

To view this administrative console page, click **Troubleshooting > Logs and trace > *server_name* > Configure HPEL Text Log**.

Enable Text Log
Enables writing log and trace records into the text log file.

Specifies that in addition to writing log and trace records in binary format, the logging system writes them in a text format as well. You can configure the text log to be formatted in either of the formats that the basic mode SystemOut.log file uses.

The text log file only contains log entries that are generated by the controller process and not those from servant or adjunct processes. As such, application log records are not written to the text log on z/OS. To view log or trace data for all application server processes, use the LogViewer command-line tool or the HPEL log viewing tool in the administrative console.

Note: All content written to the text log is also written to either the log repository or trace repository. Enabling the text log degrades performance for applications that frequently create log or trace entries.

Directory path
Specifies the directory to which log files are written.

Text log file names have the following format: TextLog_<yy.mm.dd>_<hh.mm.ss>, where “TextLog_” is a fixed prefix, <yy.mm.dd> is a date (year, month, date) of the first record in the file, and <hh.mm.ss> is the time (hour, minute, second)

Note: This path cannot be shared between servers. The server creates a file with a `.owner` extension to help detect when two or more servers happen to be trying to use the same path for HPEL output.

Enable log record buffering

Specifies that the logging system avoids writing to disk each time a log record is created. The logging system creates a buffer that can hold a number of log records, and writes the buffered events when the buffer is full. The logging system also writes the buffered events after a few seconds have passed, even if the buffer is not full.

Selecting this setting significantly improves logging performance; however, if the server stops unexpectedly, the contents might not be written to the text log file.

Note: Enable log record buffering in almost all cases. Only disable log record buffering when your server is failing unexpectedly and cannot write buffered content to disk before stopping.

Start new log file daily at <time>

Enables the logging framework to close the log file and start a new file at the specified time of day. Closing the file makes it easy to copy the file to an archive.

Note: If you want to automatically archive your log files, set up your backup program to copy files after the time you configured for new logs to be started. Configure the backup to occur at least 10 minutes after the time configured for new logs to be started to ensure that the server has closed the previous file.

Begin cleanup of oldest records

Specifies the log cleanup settings to be used to automatically purge the oldest log records, or log records that no longer fit in the configured space, from the text log directory.

Select **When log size approaches maximum** to configure automatic log file cleanup to begin when the total size of the text log files approaches the configured maximum size.

Select **When oldest records reach age limit** to configure automatic log file cleanup to begin when log content is the age limit specified.

Select **When either age or size restriction is met** to configure automatic log file cleanup to begin when either of the previous conditions is met.

Regardless of the selection chosen, text log files are deleted from the text log directory in the order in which they were written.

Log record age limit:

Specifies the lifespan, in hours, that log records can remain in the text log directory before the log records can be automatically deleted by the server. When all records in a text log file have existed longer than the age limit specified, then that file is targeted for deletion by the server.

Maximum log size:

Specifies the maximum total size, in megabytes, that the server allows the text log files to reach. When the total size of the text log files approaches this size limit, the server deletes the oldest text log files from the text log directory to make space for new log records.

Out of space action

Specifies how the server reacts to an inability to add content to the text log directory.

Select **Stop server** to specify that the server stops when the server is unable to write to the text log directory.

Select **Purge old records** to specify that the server continues to run and that the file containing the oldest log records is immediately removed when the server is unable to write to the text log directory.

Select **Stop logging** to specify that the server continues to run, but that the server cannot continue to write to the log when the server is unable to write to the text log directory.

Text Output Format
Specifies the format to use in the text log file.

Select **Basic** to specify a shorter, one-line-per-record format.

Select **Advanced** to specify a longer format using full logger name and more details about each record.

Include trace records
Specifies whether trace records are included in the text log file, as well as log records.

Save runtime changes to configuration as well
Specifies that changes are made to both the dynamic state of the running server and the server configuration, which take effect on the next restart. If this check box is not selected, the server does not copy the settings into the server configuration.

Change log and trace mode settings
Use this page to choose the mode your system uses for logging and tracing.

Note: You can only access this page when the server is configured to use HPEL log and trace mode.

To view this administrative console page, click **Troubleshooting > Logs and trace > *server_name* > Change log and trace mode**.

Cancel
Use the Cancel button to keep the server in HPEL log and trace mode.

Switch to traditional mode
Use the **Switch to Traditional Mode** button to change the log and trace mode for the server to traditional.

Important: Switching the server to traditional log and trace mode requires a server restart.

Log viewer settings

Use this page to view your High Performance Extensible Logging (HPEL) log, trace, System.out, and System.err content.

Note: You can only access this page when the server is configured to use HPEL log and trace mode.

To view this administrative console page, click **Troubleshooting** > **Logs and trace** > ***server_name*** > **View HPEL logs and trace**.

If HPEL is not enabled, you must enable it. Read about changing to HPEL logging and tracing from basic mode for more information.

You can only access this page when the server is configured to use HPEL log and trace mode. You can also view log and trace data for servers that are currently stopped as long as a node agent or administrative agent is running on the same machine and that agent is configured to use HPEL.

Log view table

Displays the log, trace, System.out, and System.err records.

The log view section displays the records. Use the First Page, Previous Page, Next Page, and Last Page buttons to move through the list of records, or specify filter criteria in the Content and Filtering Details section to limit the rows displayed. Records are always displayed in the order they were recorded by the server. By default the log view has 5 columns as listed in the following table..

Table 30. Log view table columns. The table columns provide information on the log records.

Column	Description
Time Stamp	The time when the event was recorded.
Thread ID	The identity of the thread that recorded the event in hexadecimal notation.
Logger	The logger that recorded the event.
Level	The type of event that was recorded.
Message	The message from the recorded event. If the message has a message ID the message ID is underlined. Click the message ID to get an explanation and recommended user action for the message.

To manipulate the log view, you can complete the following actions using available buttons.

Table 31. Log view table buttons. The table buttons help you configure the log settings.

Button	Resulting action
Refresh View	Clears the contents of the viewer and reinitializes the view using records from the server. Use this button to retrieve information about any additional rows created since the log viewer was started.
Show Only Selected Threads	Filters any records created by any thread other than the one selected in the selection area. Clicking on this button enables the Show All Threads button.
Show All Threads	Displays any records that were filtered when you clicked Show Only Selected Threads.. This button is only enabled when you have restricted the view using the Show Only Selected Treads button.
Select Columns	Enables you to select the columns in the viewer that you want to view.
Export	Exports logs to local workstation in any of basic, advanced, or binary (HPEL) formats
Copy to Clipboard	Copies the records that are highlighted in the selection area into the operating system clipboard.
Server Instance Information	Displays attributes for the selected server instance process. Use this table to find attributes and corresponding values for the server instance process environment. These properties are similar to the ones found in the header of basic mode logs.

Content and filtering details

Provides selection options to specify what content sources to include and what content to filter from the log view.

To change what content sources are shown in the table, or to filter what content is shown, expand the **Content and Filtering Details** section.

Server instance

Changes the server instance from which log records are retrieved.

A server instance represents a run of a server process. Each time the server is restarted, a new server instance is created. By default, the log view table shows log records generated in the most recent server instance. To select a different server start time, choose a server instance with the appropriate start timestamp. The timestamps shown represent the timestamp of the first record written to each server instance.

View contents

Controls what content sources are displayed in the log view.

System Out

Specifies that content logged to the System.out output stream is included in the log view.

System Error

Specifies that content logged to the System.err output stream is included in the log view.

Logs and trace

Specifies that log and trace records are included in the log view.

Log and trace entries can be further specified to include a minimum or maximum level. Minimum and maximum can be specified together, for example to display only a certain level of trace. If log and trace is not selected neither log nor trace records of any severity might be displayed.

Examples of log and trace filtering:

- Selecting logs and trace and clearing minimum level and maximum level fields results in the log view displaying records with any log or trace level.
- Selecting logs and trace and setting minimum level to WARNING results in the log view displaying log records with levels WARNING, FATAL, or SEVERE.
- Selecting logs and trace and setting maximum level to FINE results in the log view displaying trace records with levels FINE, FINER, or FINEST.
- Selecting logs and trace and setting minimum level to DETAIL and maximum level to AUDIT results in the log view displaying log records with levels DETAIL, CONFIG, INFO, or AUDIT.

Filtering

Controls which records are included in and excluded from the log view.

For all filters in this section, multiple entries might be specified using a colon (:) as a separator character. A limited set of regular expression characters can be used. Refer to console documentation for more details. If multiple filter settings are specified, the filter conditions must all be true for a record to be displayed in the log view.

Include loggers

Specifies the list of loggers whose records are included in the log view.

Exclude loggers

Specifies the list of loggers whose records are excluded from the log view.

Message contents

Specifies the message content that each record must contain to be included in the log view.

Event timing

Controls what records are displayed in the log view based on a start and end date and time.

From

Specifies the time of day, which the record creation time must be greater than or equal to for the record to be displayed in the log view.

Time must be specified as HH:mm:ss.SSS using the 24-hour clock. If the From value is not specified, a default value of 00:00:00.000 is used.

- HH represents the hour of the day. Valid values are from 0 to 23.
- mm represents the minute of the hour. Valid values are from 0 to 59.
- ss represents the seconds of the minute. Valid values are from 0 to 59.
- SSS represents the milliseconds of the second. Valid values are from 000 to 999.

On (first occurrence)

Specifies the date, which the record creation date must be greater than or equal to for the record to be displayed in the log view.

Until

Specifies the time of day, which the record creation time must be less than or equal to for the record to be displayed in the log view.

Time must be specified as HH:mm:ss.SSS using the 24-hour clock. If the Until value is not specified, a default value of 23:59:59.999 is used.

- HH represents the hour of the day. Valid values are from 0 to 23.
- mm represents the minute of the hour. Valid values are from 0 to 59.
- ss represents the seconds of the minute. Valid values are from 0 to 59.
- SSS represents the milliseconds of the second. Valid values are from 000 to 999.

On (second occurrence)

Specifies the date, which the record creation date must be less than or equal to for the record to be displayed in the log view.

LogViewer command-line tool

Use the LogViewer command to query the contents of the High Performance Extensible Logging (HPEL) log and trace repositories. You can also use the LogViewer command to view new log and trace repository entries as the server writes content to them.

LogViewer

The High Performance Extensible Logging (HPEL) facility writes to the log and trace repositories in a binary format. You can view, query and filter the repository using the LogViewer command. The LogViewer command provides options for quickly converting HPEL logs into a text file in various formats, including basic, advanced, and Common Base Event format. The command also provides options to make getting the data you need from the logs easier; for example, allowing you to filter what log records you want by level, logger name, or date and time.

Use the following command to view the full contents of your log and trace repositories:

Optional parameters

-repositoryDir *directory_name*
Specifies the path to the repository directory. In the case where you want to query both the log and trace data together, provide the path to the parent directory, which contains both the log data and tracedata directories. If you use the default repository location, profile_root/logs/application_server/, and run this tool from the profile bin directory, then this argument is optional. The tool checks the default location if one is not provided. If multiple application servers exist in this profile with HPEL repositories, you are prompted to select which server log and trace repository you want to view.

-outLog *file_name*
Specifies the file name you want the text output written to. If you do not provide this information, the text output is displayed on the console.

-format basic | advanced | cbe-1.0.1
Specifies the output format. Supported formats include basic, advanced, and the CBE-1.0.1 format. If you do not provide this information, the output is in basic format.

-monitor [*integer***]**
Specifies that you want the logViewer to continuously monitor the repository and output new log record entries as they are created. You can provide an optional integer argument after this parameter to specify how often you want the LogViewer tool to query the repository for new records. By default the logViewer queries the repository for new records every 5 seconds. When used with other filtering options, only those new records that match the filter criteria are displayed.

-help
Use this parameter to have the LogViewer tool list the full set of options that are available.

-startDate*date_time*
You can filter the results that are displayed from the repository by date and time. Use the startDate parameter to filter out log entries that occurred before the date or date time provided as an argument. Provide either a date or date and time, entered in the MM/dd/yy format or the MM/dd/yy H:m:s:S format.

-stopDate *date_time*
Use this parameter to filter out log entries that occurred after the specified date or date time. Provide the argument in the same format as the -startDate option.

-level *level_name*
Specifies that you want the tool to only display those log events which match the level name you provide as an argument. Valid values for the level name are FINEST, FINER, FINE, DETAIL, CONFIG, INFO, AUDIT, WARNING, SEVERE, FATAL.

-minLevel *level_name*
Specifies that you want the tool to only display records which are at or above the specified level. Valid values for the level name are FINEST, FINER, FINE, DETAIL, CONFIG, INFO, AUDIT, WARNING, SEVERE, FATAL.

-maxLevel *level_name*
Specifies that you want the tool to only display records that are at or below the specified level. Valid values for the level name are FINEST, FINER, FINE, DETAIL, CONFIG, INFO, AUDIT, WARNING, SEVERE, FATAL.

-includeLoggers *logger_name*
When this option is used, only log events from the specified loggers are included in the LogViewer output. Separate multiple entries with a comma. The * symbol can be used as a wild card to include all loggers below a parent logger. When used in combination with the -excludedLoggers option, the more specific match determines if the log event is included or excluded.

-excludeLoggers *logger_name*
Use this option to exclude log events from the specified loggers in the LogViewer output. Separate multiple entries with a comma. The * symbol can be used as a wildcard to include all loggers below a parent logger. When used in combination with the -includeLoggers option, the more specific match determines if the log event is included or excluded.

-thread *thread_id*
Use this option to restrict LogViewer output to only those log events from a specific thread. Any log messages that were not created by the thread ID provided as an argument to this option are not displayed. Specify the thread ID in hex format.

-extractToNewRepository*directory_name*
This option redirects log and trace records from a binary repository to a new binary repository at the location that you specify. You can use this option with other filtering options to get a subset of log and trace records into the new repository. This option uses the directory path where the new repository must be written as an argument. Therefore, the directory must be empty. If the directory does not exist, the directory is created. However, errors that occur during the directory creation might create extraneous directories.

-listInstances
Use this option to list the IDs of available server process instances that are available to use with the -instance option. After running LogViewer with the -listInstances option, you can then use the -instance option to invoke LogViewer with one of the server process instance IDs as an argument. Since this option does not process any log or trace records, all other options are ignored when you specify this option.

-instance *instance_id*
Use this option to retrieve the log and trace data for a given server process instance by providing the server instance ID. Run LogViewer, along with the -listInstances option, before you use this option to obtain a valid instance ID. This option is required when viewing logs and trace from an environment that contains subprocesses, such as the z/OS operating system.

If this option is combined with -latestInstance, -instance is ignored.

-latestInstance
Use this option to retrieve the log and trace data from the most recent server instance. If this option is used with the -instance option, the -instance option is ignored.

-message *match_string*
Use this option to retrieve only log or trace data with a message field that matches the requested text.

-includeExtensions name[=*value***][,name[=***value***]]***
Use this option to retrieve the log and trace data with an extension name that matches the requested name, and an extension value that matches the requested value. You can also use this option to retrieve the log and trace data with an extension name that matches the requested name, and an extension value that matches any value, if you omit the =*value* part of the option.

Any extension name shown in the advanced format can be used. Note that 'source', 'class', and 'method' are not stored in the log/trace repositories as extensions, and so cannot be filtered on with this option.

Separate multiple name=*value* arguments with a comma. Specify '==' (two equals signs) in place of '=' (one equals sign) in cases where the name or value must contain an equal sign. Specify ',,' (two commas) in place of ',' (one comma) in cases where the name or value must contain a comma.

-encoding *character_set*
Specifies the character set that the LogViewer command will use for text output.

Filtering considerations

Be aware of LogViewer filtering optimizations. The LogViewer tool is able to filter log and trace data most efficiently when used with the following filter options:

- startDate
- stopDate
- thread
- level
- minLevel
- maxLevel

Example usage

See the following examples of LogViewer commands on UNIX-based systems. The examples show how to run LogViewer from the profile bin directory where the repositoryDir parameter is not required.

- Write all records in the default repository between July 19th, 2009 and August 2nd, 2009 to a file called /tmp/promo.logs.

```
logViewer.sh -outLog /tmp/promo.logs -startDate 07/19/2009 -stopDate 08/02/2009
```

- Display new records whose specified level is WARNING or higher using the advanced format as the server writes them to the log repository.

```
logViewer.sh -monitor -minLevel WARNING -format advanced
```

- Write only those log messages that were written to the error stream of a specific repository to a file called logged_errors.txt.

```
logViewer.sh -repositoryDir /apps/server1/logs -includeLoggers SystemErr -outLog logged_errors.txt
```

- View events from the default repository that occurred before September 14th, 2009 4:28 PM eastern daylight time.

```
logViewer.sh -stopDate "09/14/2009 16:28:00:000 EDT"
```

- Write events from the default repository that contain a 'thread' extension with value 'WebContainer : 6'

```
logViewer.sh -includeExtensions thread="WebContainer : 6" -format advanced
```

- Write events from the default repository that were a part of the request with requestID a856cb2c-79ed-4d62-a3cf-a9908b2db07b.

```
logViewer.sh -includeExtensions requestID=a856cb2c-79ed-4d62-a3cf-a9908b2db07b
```

- Write events from the default repository that were created on a thread servicing the PlantsByWebSphere application.

```
logViewer.sh -includeExtensions appName=PlantsByWebSphere
```

On z/OS operating systems where multiple processes exist, you must provide the instance ID to identify which process you want to view logs and trace from. The instance ID of a controller is represented by a numeric value while the instance ID of a servant is represented by a combination of a numeric value, job name, job ID, and process ID. To obtain a list of valid instance IDs, run LogViewer with the -listInstances option.

- Invoke LogViewer with the -listInstances option; for example:

```
logViewer.sh -listInstances
```

  The following example is a list of instance IDs from one controller and three servants:

```
Instance ID            Start Date
1280334046      5/10/10 18:53:12:770 GMT
1280334046/000001BC00000002_BBOS1S_STC003119  5/10/10 18:53:39:220 GMT
1280334046/000001B400000002_BBOS1S_STC003120  5/10/10 18:54:44:339 GMT
1280334046/000001C000000001_BBOS1S_STC003121  5/10/10 18:55:43:520 GMT
```

- Invoke LogViewer with the -instance option using one of the instance IDs from the previous example. The ID type is a controller; for example:

```
logViewer.sh -instance 1280334046
```

- Invoke LogViewer with -instance option for a servant instance; for example:

Monitoring application logging using JMX notifications

Java developers can create programs to monitor application server logs using JMX notifications.

About this task

The most common log message listeners are written in Java, and connect to the deployment manager or an application server using SOAP. Use this topic to build a Java client that listens for log events.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Note: Be careful when adding listeners to servers with high logging volume as JMX notifications can slow down your server.

Procedure

1. Import the necessary packages. You will typically need the following import statements at the beginning of your Java program:

```
import javax.management.Notification;
import javax.management.NotificationListener;
import javax.management.ObjectName;
import javax.management.InstanceNotFoundException;
import javax.management.MalformedObjectNameException;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.exception.ConnectorException;
```

   Additionally, to handle the messages, and the types returned from the calls in subsequent steps you will need the following import statements.

```
import java.util.Iterator;
import java.util.Properties;
import java.util.Set;
import com.ibm.websphere.ras.RasMessage;
```

2. Create a Java class that implements the NotificationListener interface.
3. Implement the handleNotification method. The following example is a sample that writes the message text to the Java console:

```
public void handleNotification(Notification notification, Object handback) {
        RasMessage rasMessage = (RasMessage)notification.getUserData() ;
        System.out.println("Localized message: " + rasMessage.getLocalizedMessage(null));
}
```

4. Connect to the SOAP port of the server whose JMX MBeans you want to monitor. The following code creates a SOAP-connected AdminClient object with a specified host and a specified port:

```
        AdminClient adminClient = null ;
        String hostName = "someHostName";
        String soapPort = "8880";

        Properties connectProps = new Properties();
        connectProps.setProperty(AdminClient.CONNECTOR_TYPE, "SOAP");
        connectProps.setProperty(AdminClient.CONNECTOR_HOST, hostName);
        connectProps.setProperty(AdminClient.CONNECTOR_PORT, soapPort);
```

```
try {
    adminClient = AdminClientFactory.createAdminClient(connectProps);
} catch (ConnectorException e) {
    // error handling code
}
```

5. Retrieve the MBean object name for the RasLoggingService MBean. The following code retrieves the RasLoggingService MBean object name:

```
String queryString = "WebSphere:cell="+cellName+",node="+nodeName+",process="+serverName+",
type=RasLoggingService,*" ;
Set<ObjectName> objectMBeans = null;
try {
    ObjectName queryName = new ObjectName(queryString);
    objectMBeans = (Set<ObjectName>)adminClient.queryNames(queryName, null);
    if (objectMBeans.size() > 1) {
        // error handling code to deal with the case where we get more than one name returned.
    }
} catch (MalformedObjectNameException e) {
    // error handling code
} catch (ConnectorException e) {
    // error handling code
}

if (objectMBeans.isEmpty()) {
    // error handling code to deal with the case where the MBean is not found
}

Iterator<ObjectName> objectNames = objectMBeans.iterator() ;
ObjectName objectName = objectNames.next() ;
```

6. Add the notification listener. This sample code adds a notification listener, waits for 60 seconds while it processes notifications, then removes the notification listener. A listener can stay connected as long as needed.

```
try {
    adminClient.addNotificationListener(objectName, this, null, null);
    Thread.sleep(60 * 1000) ;
    adminClient.removeNotificationListener(objectName, this) ;
} catch (InstanceNotFoundException e) {
    // error handling code
} catch (ConnectorException e) {
    // error handling code
} catch (Exception e) {
    // error handling code
}
```

7. Add the necessary jar to your classpath. Add the admin client jar file to your classpath to be able to compile and run your code. The admin client jar file is in the <install_root>/runtimes directory.

Results

You have created a Java program that can listen to, and take actions as a result of log event notifications from an application server.

Chapter 7. Using Cross-Component Trace to troubleshoot applications

You can use Cross-Component Trace (XCT) to help diagnose problems in WebSphere Application Server.

About this task

Administrators using WebSphere Application Server need to use log and trace files to determine whether their applications and the server are running correctly.

Depending on the nature of your applications, multiple threads within an application server may be used to handle requests, such as HTTP requests or JMS requests. Some requests may be handled by more than one application server, such as when one application server makes a request to another application server for a web services request.

You can use XCT to augment your log and trace files with correlation information. This correlation information clarifies which threads and which application server processes participated in the handling of each request.

Procedure

1. Enable XCT if you have not done so already.
   a. Log on to the administrative console.
   b. If using an admin agent topology, select a node that you want to manage and navigate to it.
   c. From the navigation section in the console, choose **Troubleshooting > Logs and trace**.
   d. Select the server that you want to enable XCT for.
   e. Choose **Change log detail levels**.
   f. Select the **Configuration** tab if you want to make static configuration changes, or the **Runtime** tab if you want to make changes to the runtime state of the server.
   g. Check **Enable log and trace correlation** checkbox.
   h. Select **Include request IDs in log and trace records**, **Include request IDs in log and trace records and correlation log records**, or **Include request IDs in log and trace records, create correlation log records, and capture data snapshots** as appropriate.
   i. Click **OK**.
   j. If you made your changes on the Configuration tab, save them and restart the application server.

   Note: Enable XCT to include request IDs in log and trace files when you want to see which log and trace entries, in all threads and application server processes, are related to the same request. Request IDs are only recorded when using HPEL log and trace mode and can be seen or used for filtering using the logViewer command.

   Note: Enable XCT to create correlation log records when you want to log how requests branch between threads and processes, and see extra information about each request. Enabling XCT to create correlation log records might have a significant performance impact on your system, so is best suited to test and development environments.

   Note: Enable XCT to capture data snapshots when you want to store entire request and response bodies to the file system. Enabling XCT to capture data snapshots might have a significant performance impact on your system, so is best suited to test and development environments. XCT captures data snapshots for message requests and responses handled by the SIBus.

   Note: Data snapshots are captured and written to the *$SERVER_LOG_ROOT*/snapdata directory. The application server does not automatically clean up files from this directory. You will need to

delete the files from this directory periodically when data snapshot capturing is enabled. Data snapshots store entire request and response contents and may include sensitive information. This option might not be appropriate for use in production environments.

2. Use XCT request ID information to track requests.
   a. Ensure you are using High Performance Extensible Logging (HPEL) log and trace mode, since basic mode log and trace does not store request IDs. Read the Changing from basic mode to HPEL logging and tracing topic for more information.
   b. Enable XCT if you have not done so already.
   c. Filter your logs to look for important information, such as errors or warnings using the HPEL LogViewer command-line tool. Output your logs using advanced format so that you can see the request ID information in the logs. For example, use the command, as follows:

      ```
      logViewer.sh -minLevel WARNING -format advanced -instance <instanceid>
      ```

   d. When you have found log entries that are of interest to you, note the request ID associated with those entries.
   e. Filter your logs by request ID using the HPEL LogViewer command-line tool and using the request IDs you noted in the previous step, as follows:

      ```
      logViewer.sh -includeExtensions requestID=<requestID> -instance  <instanceid>
      ```

      Read the LogViewer command-line tool topic for more information.
3. Use XCT correlation log records to determine the hierarchy of call chains.
   a. Enable XCT if you have not done so already. Select **Include request IDs in log and trace records and correlation log records** to ensure that XCT creates correlation log records.
   b. If you are using HPEL, convert your logs to text using the LogViewer command-line tool. For example, use the command, as follows:

      ```
      logViewer.sh -instance <instanceid> -outLog myLog.txt
      ```

      Read the LogViewer command-line tool topic for more information.
   c. Use the XCT Log Viewer tool available with the IBM Support Assistant to load log and trace files from all application servers involved in handling your requests and see the hierarchy of request call chains. Read the Using IBM Support Assistant topic for more information.

Results

The server is now configured to use XCT.

Cross Component Trace (XCT)

Cross Component Trace (XCT) annotates the logs so that log entries that are related to a request that is serviced by more than one thread, process, or even server are identified as belonging to the same unit of work.

Overview

Note: XCT helps identify the root cause of problems across components, which provides the following benefits:

- Enables administrators and support teams to follow the flow of a request from end-to-end as it traverses thread or process boundaries, or travels between stack products and WebSphere Application Server.
- Helps to resolve questions about which component is responsible for a request that fails.

Administration of XCT

XCT is a capability built into the WebSphere Application Server log and trace framework. When enabled, XCT annotates the logs so that log entries that are related to a request that is serviced by more than one thread, process, or even server are identified as belonging to the same unit of work. Applications built using distributed architectures, such as Service Oriented Architecture, will benefit from XCT, since XCT helps facilitate problem determination across multiple services on different systems.

XCT different modes:

- Fully disabled.
- With XCT request IDs added to existing log and trace records.
- With XCT request IDs added to existing log and trace records and XCT log records added to log files.
- With XCT request IDs added to existing log and trace records, XCT log records added to log files, and data snapshots captured.

XCT request IDs:

- XCT request IDs are identifiers added to log and trace records produced by the server when the server is configured to use High Performance Extensible Logging (HPEL).
- XCT adds the same request ID to every log or trace record as long as the log or trace record is a part of the same request, regardless of which thread or Java virtual machine (JVM) produces the log or trace entry.
- When XCT is used with the HPEL log and trace infrastructure, you can view request IDs with the logViewer tool when logs are output in advanced format.

The following is an example of a log record with an XCT request ID in the log file (shown rendered in advanced format):

```
[3/18/11 14:50:17:391 EDT] 00000018 W UOW= source=com.ibm.somelogger.QuickLogTest org= prod= component=
 thread=[WebContainer : 1] requestID=BJrcVPo+Yk4-AAAAAAA8zAA hello world
```

Note that the request ID is shown previously on a separate line, but in the log files it is actually on the same line as the rest of the log record header.

XCT log records:

- XCT log records are typically added to the logs to:
  - demarcate the beginning and ending of work for a particular request on a particular thread.
  - demarcate when work is about to be transferred to another thread or process, or to indicate when work returned from another thread or process.
  - demarcate when work moves from major component to major component, even if work continues on the same thread; for example to show transfer of control from application server code to application code.

  The following is an example of an XCT log record in the log file:

  ```
  [3/18/11 14:50:17:391 EDT] 00000031 XCT I BEGIN BJrcVPo+Yk4-AAAAAAA8zAA 00000000000-ccccccccc2
   HTTPCF(OutboundRequest /index.html RemoteAddress(127.0.0.1) RequestContext(36001645))
  ```
- XCT log records are composed of:
  - XCT type (BEGIN / END)
  - XCT parent correlator ID (for example, 00000000000-ccccccccc2)
  - XCT current correlator ID (for example, BJrcVPo+Yk4-AAAAAAA8zAA)
  - XCT annotations (for example, HTTPCF(OutboundRequest /index.html RemoteAddress(127.0.0.1) RequestContext(36001645))

XCT tools:

- The HPEL logViewer tool is able to filter log and trace records by request ID.

- Tools, such as the XCT Log Viewer, can also take advantage of XCT log records or XCT request IDs, or both, when rendering log and trace content. The XCT Log Viewer is available as a tool add-on for the IBM Support Assistant.

XCT configuration:

You can configure XCT using wsadmin scripting.

Configuring XCT with wsadmin scripting

You can configure Cross-Component Trace (XCT) using wsadmin scripting. Use the examples in this topic as a guide to build your own wsadmin scripts.

About this task

XCT annotates the logs so that log entries that are related to a request that is serviced by more than one thread, process, or even server are identified as belonging to the same unit of work. You can configure XCT using the administrative console, or using wsadmin scripting. The examples in this topic show how to configure XCT using wsadmin. If you complete this task using the deployment manager, then you might need to synchronize the node agent on the target node and restart the server before configuration changes take effect.

Table 32. Variable Names. The table applies to all examples in this topic. All examples use the Jython scripting language.

Variable	Description
myCell	The name of the cell
myNode	The host name of the node
myServer	The name of the server

Procedure

- Use the AdminConfig object to configure XCT.

  Changes you make using the AdminConfig object take effect the next time you start the server.

  1. Enable XCT for the server.

     By default, XCT is disabled for the server. The following example shows how to enable XCT for the server when your server is using High Performance Extensible Logging (HPEL) log and trace mode. Specify RASLoggingService instead of HighPerformanceExtensibleLogging when your server is using basic log and trace mode. Specify false instead of true to disable XCT.

     ```
     # get rid of existing property if already present
     configId = AdminConfig.getid("/Cell:myCell/Node:myNode
      /Server:myServer/HighPerformanceExtensibleLogging:
      /Property:com.ibm.websphere.logging.enableCorrelation")if (len(configId) > 0):
         AdminConfig.remove(configId)
     # add new property
     LoggingService = AdminConfig.getid("/Cell:myCell/Node:myNode
     /Server:myServer/HighPerformanceExtensibleLogging:/")
     AdminConfig.create("Property", LoggingService, [["name", "com.ibm.websphere.logging.enableCorrelation"],["value", "true"]])

     AdminConfig.save()
     ```

  2. Change the log setting for XCT .

     By default, the XCT log setting includes request IDs in log and trace files. The following example shows how to make XCT both include request IDs in log and trace files and log XCT log records. Specify REQUEST ID instead of LOG to make XCT only include request IDs in log and trace files. Specify DATA_SNAPSHOT instead of LOG to make XCT include request IDs in log and trace files, log XCT log records, and capture data snapshots. Specify RASLoggingService instead of HighPerformanceExtensibleLogging when your server is using basic log and trace mode.

```
# get rid of existing property if already present
configId = AdminConfig.getid("/Cell:myCell/Node:myNode
/Server:myServer/HighPerformanceExtensibleLogging:/Property:com.ibm.websphere
 .logging.correlationLevel") if (len(configId) > 0): AdminConfig.remove(configId)
# add new property
LoggingService = AdminConfig.getid("/Cell:myCell/Node:myNode
Server:myServer/HighPerformanceExtensibleLogging:/")
AdminConfig.create("Property", LoggingService, [["name", "com.ibm.websphere.logging.correlationLevel"],["value", "LOG"]])

AdminConfig.save()
```

Note: Enable XCT to include request IDs in log and trace files when you want to see which log and trace entries, in all threads and application server processes, are related to the same request. Request IDs are only recorded when using HPEL log and trace mode and can be seen or used for filtering using the logViewer command.

Note: Enable XCT to create correlation log records when you want to log how requests branch between threads and processes, and see extra information about each request. Enabling XCT to create correlation log records might have a significant performance impact on your system, so is best suited to test and development environments.

Note: Enable XCT to capture data snapshots when you want to store entire request and response bodies to the file system. Enabling XCT to capture data snapshots might have a significant performance impact on your system, so is best suited to test and development environments. XCT captures data snapshots for message requests and responses handled by the SIBus.

Note: Data snapshots are captured and written to the *$SERVER_LOG_ROOT*/snapdata directory. The application server does not automatically clean up files from this directory. You will need to delete the files from this directory periodically when data snapshot capturing is enabled. Data snapshots store entire request and response contents and may include sensitive information. This option might not be appropriate for use in production environments.

- Use the AdminControl object to configure XCT. Changes you make using the AdminControl object take effect immediately.
  1. Enable XCT for the server.

     By default, XCT is disabled for the server. The following example shows how to enable XCT for the server when your server is using HPEL log and trace mode. Ensure the server is running and specify RasLoggingService instead of HPELControlService when your server is using basic log and trace mode. Specify false instead of true to disable XCT.

     ```
     LoggingMBean = AdminControl.queryNames('cell=myCell,node=myNode,
     type=HPELControlService,process=myServer,*')
     AdminControl.setAttribute(LoggingMBean, "correlationEnabled", "true")
     ```

  2. Change the log setting for XCT .

     By default, the XCT log setting includes request IDs in log and trace files. The following example shows how to make XCT both include request IDs in log and trace files and log XCT log records. Specify REQUEST ID instead of LOG to make XCT only include request IDs in log and trace files. Specify DATA_SNAPSHOT instead of LOG to make XCT include request IDs in log and trace files, log XCT log records, and capture data snapshots. Ensure the server is running and specify RasLoggingService instead of HPELControlService when your server is using basic log and trace mode.

     ```
     LoggingMBean = AdminControl.queryNames('cell=myCell,node=myNode,type=HPELControlService,process=myServer,*')
     AdminControl.setAttribute(LoggingMBean, "xctLevel", "LOG")
     ```

     Note: Enable XCT to include request IDs in log and trace files when you want to see which log and trace entries, in all threads and application server processes, are related to the same request. Request IDs are only recorded when using HPEL log and trace mode and can be seen or used for filtering using the logViewer command.

Note: Enable XCT to create correlation log records when you want to log how requests branch between threads and processes, and see extra information about each request. Enabling XCT to create correlation log records might have a significant performance impact on your system, so is best suited to test and development environments.

Note: Enable XCT to capture data snapshots when you want to store entire request and response bodies to the file system. Enabling XCT to capture data snapshots might have a significant performance impact on your system, so is best suited to test and development environments. XCT captures data snapshots for message requests and responses handled by the SIBus.

Note: Data snapshots are captured and written to the $*SERVER_LOG_ROOT*/snapdata directory. The application server does not automatically clean up files from this directory. You will need to delete the files from this directory periodically when data snapshot capturing is enabled. Data snapshots store entire request and response contents and may include sensitive information. This option might not be appropriate for use in production environments.

Results

XCT is now configured. If you made changes with the AdminConfig command, restart the server to make the changes take effect.

Chapter 8. Using sensitive log and trace guard

You can protect information with the sensitive log and trace guard. The sensitive log and trace guard prevents loggers from writing sensitive information in your log and trace files.

Sensitive log and trace guard

The sensitive log and trace guard is a feature that helps administrators prevent sensitive information from being exposed in log and trace files.

The sensitive log and trace guard uses an internal list of allowable levels for sensitive loggers which specifies the lowest level at which listed loggers can generate log or trace data without containing potentially sensitive data. You can also add your own loggers to the list that the sensitive log and trace guard will block.

An example is as follows: If a servlet writes URL request parameters verbatim to logger com.xyz.SomeLogger at level Level.FINE, and these request parameters could contain information such as credit card numbers or passwords, then you should add an entry to the sensitive logger list to allow only levels higher than Level.FINE to be logged - com.xyz.SomeLogger=CONFIG.

When the server initializes the log and trace system, or when you attempt to change the log detail level for a server, the list of allowable levels for sensitive loggers is compared to the stated log detail level. Any attempt to enable logging or tracing that is in conflict with entries in the list is overridden. In cases where the same loggers are specified multiple times in the list the most restrictive entry is used. For example, if the list of allowable levels for sensitive loggers contains a.b.*=INFO and a.b.*=FINE, then the a.b.* loggers are restricted to only being able to log at levels INFO and higher. The following table provides examples of how lists of allowable levels for sensitive loggers modify specified log detail level settings to determine effective log detail level settings:

Table 33. Restriction list. Examples of how sensitive log and trace guard settings affect log detail level settings

Log detail level setting	List of allowable levels for sensitive loggers	Resulting effective log detail level setting
a.b.*=SEVERE	a.b.*=FINE	*=INFO:a.b.*=SEVERE
a.b.*=SEVERE	a.b.*=SEVERE	*=INFO:a.b.*=SEVERE
a.b.*=FINE	a.b.*=FATAL	*=INFO:a.b.*=FATAL
a.*=SEVERE	a.b.*=FINE	*=INFO:a.*=SEVERE
a.*=SEVERE	a.b.*=SEVERE	*=INFO:a.*=SEVERE
a.*=FINE	a.b.*=FATAL	*=INFO:a.*=FINE:a.b.*=FATAL
a.b.*=SEVERE	a.*=FINE	*=INFO:a.b.*=SEVERE
a.b.*=SEVERE	a.*=SEVERE	*=INFO:a.b.*=SEVERE
a.b.*=FINE	a.*=FATAL	*=INFO:a.b.*=FATAL
a.b.*=FINE	*=SEVERE	*=SEVERE:a.b.*=SEVERE
a.b.*=FINE	*=FINE	*=INFO:a.b.*=FINE
a.b.*=FINE	*=FINEST	*=INFO:a.b.*=FINE
a.b.*=FINE	x.y.z.*=SEVERE	*=INFO:a.b.*=FINE:x.y.z.*=SEVERE
a.b.*=FINE	x.y.z.*=FINE	*=INFO:a.b.*=FINE
a.b.*=FINE	x.y.z.*=FINEST	*=INFO:a.b.*=FINE
a.b.*=FINE	*=WARNING:x.y.z.*=SEVERE	*=WARNING:a.b.*=WARNING:x.y.z.*=SEVERE
a.b.*=FINE	*=WARNING:*=SEVERE:x.y.z.*=SEVERE	*=SEVERE:a.b.*=SEVERE

By using this log and trace guard, you can prevent loggers from logging at levels which might expose sensitive information.

The product is preconfigured with a known list of loggers to restrict, however you might find that further restrictions are required. .

Enabling and disabling sensitive log and trace guard

You can either enable or disable the sensitive log and trace guard to help control whether loggers write sensitive information in your log and trace files.

About this task

Administrators using WebSphere Application Server can prevent sensitive information, such as data provided from users in HTTP requests, from being written in log and trace files. In some cases, when having access to private data can help with debugging, you might want to disable sensitive log and trace guard. For example, you might see that a credit card number that was entered in a web form did not have the required number of digits.

Sensitive log and trace guard works by preventing administrators from enabling certain loggers to levels at which they are known to log or trace sensitive information.

Use the administrative console to enable or disable the sensitive log and trace guard.

Procedure

1. Log on to the administrative console.
2. If you are using an administrative agent topology, then select a node that you want to manage, and navigate to it.
3. From the navigation section in the console, choose **Troubleshooting** > **Logs and trace**.
4. Select the server that you want to enable or disable with sensitive log and trace guard.
5. Click **Change log detail levels**.
6. Select the **Disable logging and tracing of potentially sensitive data** check box to enable sensitive log and trace guard. To disable sensitive log and trace guard, clear the **Disable logging and tracing of potentially sensitive data** check box.
7. Click **OK**
8. Save the changes.

Results

After you enable sensitive log and trace guard, the server is now configured to prevent known sensitive loggers from writing sensitive content to the log and trace files. After you disable sensitive log and trace guard, the server is now configured to allow known sensitive loggers to write sensitive content to the log and trace files. If you completed these steps using the deployment manager, you might need to synchronize the node agent on the target node before restarting the server.

Maintaining sensitive log and trace guard lists

The sensitive log and trace guard relies on lists which declare which loggers can potentially log or trace sensitive information, and the levels at which the sensitive information would be logged. You can extend the default list of loggers and their corresponding levels in cases where you find sensitive information in your log or trace that you want to block from being logged or traced in the future.

Before you begin

Read about log level settings for information about enabling the Sensitive Log and Trace Guard.

About this task

The application server has a private default list of sensitive loggers and their corresponding levels which it will block whenever the sensitive log and trace guard feature is enabled. The application server also provides a sensitive log and trace guard property file, and a sensitive log and trace guard API that you can use to declare new logger restrictions if you discover other loggers which log or trace sensitive information.

Note: If you attempt to add loggers to the sensitive log and trace guard list that have already been declared, the sensitive log and trace guard will use the more restrictive logger setting of the already declared and newly specified levels. For example, if the server is already configured to only allow logger com.xyz.SomeLogger to log at level FINE, and you attempt to declare that the same logger should only be allowed to log at level FINEST, the server will ignore the update, but if you attempt to declare that the same logger should only be allowed to log at level INFO, then the server will reconfigure the sensitive log and trace guard to use level INFO for that logger.

Procedure

- You can use a properties file to declare new logger restrictions. This file is in the cell-scoped configuration for each profile. The name is:

  `<profileHome>/config/cells/<cellname>/ras.rawtracelist.properties`

  This file contains documentation and syntax samples, but contains no actual entries. If you edit this file on the deployment manager the file is automatically synchronized with all nodes in the cell. If you edit this file on a specific node, it will be replaced the next time the file is synchronized with the deployment manager. Thus, it is best to maintain the list at the deployment manager.
- You can use the com.ibm.websphere.logging.RawTraceList API to declare new logger restrictions This API allows you to add individual entries or an array of entries (using the PatternLevel object in the same package). It also allows passing in an input stream in the same format as the properties file.

Chapter 9. Diagnosing problems (using diagnosis tools)

Various diagnosis tools are provided to help you determine the source and impact of problems occurring in your application serving environment.

About this task

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 non-optimized performance, problem determination focuses on finding solutions to functional problems.

Procedure

1. If deploying or running an application results in exceptions such as ClassNotFoundException, use the Class Loader Viewer to diagnose problems with class loaders.
2. If you already have an error message and want to quickly look up its explanation and recommended response, look up the message by expanding the Messages section of the Information Center under Reference > Messages.
3. If you are using the basic or traditional log and trace infrastructure, and need help finding error and warning messages, interpreting messages, and configuring log files, see Chapter 10, “Using basic or traditional message logs to troubleshoot applications,” on page 149.
4. If you are using the High Performance Extensible Logging (HPEL) log and trace infrastructure, and need help finding error and warning messages, interpreting messages, and configuring log files, see Chapter 6, “Using High Performance Extensible Logging to troubleshoot applications,” on page 107.
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 viewing diagnostic information like dumps, error logs and CTRACE information, see Viewing diagnostic information
7. To learn how to work with Diagnostic Providers, see Working with Diagnostic Providers..
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 website, or to obtain technical support from IBM, see Obtaining help from IBM.
9. The IBM developer kits: Diagnosis documentation 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 WebSphere Application Server Product support page. For last minute updates, limitations, and known problems, refer to the Release notes section.
11. Before opening a PMR, there is MustGather information that you need to collect to send to IBM Support. This information assists IBM Support in identifying and resolving your problem. You can use the IBM Support Assistant Data Collector to automatically collect the MustGather information from your system.
12. To automatically gather Must gather type information from your system, see IBM Support Assistant Data Collector.

Chapter 10. Using basic or traditional message logs to troubleshoot applications

WebSphere Application Server can write system messages to several general purpose logs, including JVM, process, and IBM service logs, which can be examined for problem determination.

Before you begin

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 and `System.err` log can be used for problem determination. The `System.err` log contains exception stack trace information that is useful when performing problem analysis.

Because each application server represents a JVM, there is one set of JVM logs for each application server and all of its applications located by default in the following directory:

- *install_root*/profiles/*profile_name*/logs/*server_name*

In the case of a WebSphere Application Server, Network Deployment configuration, JVM logs are also created for the deployment manager and each administrative agent because they also represent JVMs.

There is one set of `STDOUT` and `STDERR` log streams for each application server and all of its applications. JVM logs are also created for the deployment manager and each administrative agent because they also represent JVMs.

The process logs are created by redirecting the `STDOUT` and `STDERR` streams of the process to independent log files. Native code, including the Java virtual machine (JVM) itself, writes to these files. As a general rule, WebSphere Application Server does not write to these files. However, these logs can contain information relating to problems in native code or diagnostic information written by the JVM.

As with JVM logs, there is a set of process logs for each application server, since each JVM is an operating system process. For WebSphere Application Server, Network Deployment configuration, a set of process logs is created for the deployment manager and each administrative agent.

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

In addition to these general purpose logs, WebSphere Application Server contains other specialized logs that are specific to a particular component or activity. For example, the HTTP server plug-in maintains a special log. Normally, these logs are not of interest, but you might be instructed to examine one or more of these logs while performing specific problem determination procedures. For details on how and when to view the plug-in log, see the Accessing a web resource through the application server and bypassing the HTTP server subsection of the A web resource does not display topic.

Note: The `System.out` and `STDOUT` streams are redirected to the `SYSPRINT` ddname under z/OS. The `System.err` and `STDERR` streams are redirected to the `SYSOUT` ddname under z/OS. By default, the

WebSphere Application Server for z/OS cataloged procedures associate these ddnames with print (SYSOUT=*) data sets, causing message logs to go into WebSphere Application Server job output. Job output can be viewed with the Spool Display and Search Facility (SDSF) or equivalent software.

About this task

Sometimes server and application problems can be diagnosed by examining log output from the WebSphere Application Server.

Procedure

Determine which type of logs you would like to implement:

- JVM logs
- IBM service logs

Example

How to direct SYSPRINT and SYSOUT output to an HFS file.

If you are familiar with UNIX or Windows environments, you might be reluctant to use the facilities of SDSF (or IOF) to view the SYSPRINT and SYSOUT output from servants. If you would rather use a familiar editor (such as vi) in a Telnet session to view your output, it is possible to redirect the SYSPRINT and SYSOUT outputs to files in an HFS.

The JCL example below shows how to modify the SYSPRINT DD card in your startup procedure to redirect the output to an HFS file. The old SYSPRINT DD card has been commented out by preceding it with /*, and a new SYSPRINT DD card points to a file in the "/myDir/myServer" directory, in this case named `was.log.d&LYYMMDD..t&LHHMMSS.log`. The extra period between the date and time variables is not a typographical error, but rather an instance of JCL syntax that is necessary to terminate the first variable. &LYYMMDD will be replaced with the local date in YYMMDD format and &LHHMMSS will be replaced by the local time in HHMMSS format. The PATHMODE subparameter sets the file mode to 775 and the PATHOPTS subparameter OWRONLY opens the file for WRITE access. The sub-parameter OCREAT indicates that if the file does not already exist, create it.

You can modify the SYSPRINT DD card in either your Servant or Controller startup procedure. In addition, the SYSOUT DD card can be modified in the same way to redirect the SYSOUT output.

```
//*YSPRINT  DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//SYSPRINT  DD PATHMODE=(SIRWXU,SIRWXG,SIROTH),
//   PATHOPTS=(OWRONLY,OCREAT),
//   PATH='/myDir/myServer/was.log.d&LYYMMDD..t&LHHMMSS'
```

Note: If you try to direct the output for multiple streams to the same file, such as setting both DEFALTDD and HRDCPYDD variables, the allocation for the HRDCPYDD file fails and output is sent to the default location (JOBLOG/SYSLOG).

Viewing JVM logs

The Java virtual machine (JVM) logs are written as plain text files.

About this task

The `SystemOut.log` and `SystemErr.log` JVM logs are located in the job logs of the application server.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace

infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

JVM log interpretation

View the JVM log files to determine problems within application environments.

The JVM logs contain print data written by applications. The application can write this data directly in the form of `System.out.print()`, `System.err.print()`, or other method calls. The application can also write data indirectly by calling a JVM function, such as an `Exception.printStackTrace()`. In addition, the `System.out` JVM log contains system messages written by the WebSphere Application Server.

If you allow the application server to format the application data, it is printed in the normal z/OS trace format. If you do not allow the application server to format the application data, the raw text is printed, which is much harder to analyze.

Note: You can configure the thread ID that is used in the log and trace files of WebSphere Application Server Version 8.5 as either the hexadecimal representation of the thread ID from java.util.logging.LogRecord, or the hexadecimal representation of the thread ID from java.lang.Thread. Set the Java system property, com.ibm.websphere.logging.useJULThreadID. to `true` to have the thread ID match the java.util.logging.LogRecord thread ID. Set the system property to `false` to have the thread ID match the java.lang.Thread thread ID. If the system property is not specified, the thread ID matches the java.lang.Thread thread ID.

Format of an error log entry

```
1  2005/03/02 17:31:17.641 01 t=8FB718 c=UNK key=S2 (13007002)
2  ThreadId: 0000004e
3  FunctionName: com.ibm.ws.sm.workspace.impl.WorkSpaceManagerImpl
4  SourceId: com.ibm.ws.sm.workspace.impl.WorkSpaceManagerImpl
5  Category: AUDIT
6  ExtendedMessage: BBOO0222I: WKSP0023I: Workspace configuration consistency check is disabled.
```

Table 34. Parts of a log stream record. The following table explains the error log entry previously mentioned.

Line number	Component	Description
1	2005/03/02 17:31:17.641 01	Date / timestamp / 2-digit record version number
1	t=8FB718	MVS TCB (thread) Address
1	c=UNK	Request correlation information
1	key=S2	State/Key (S=Supervisor,P=Problem)
1	(13007002)	Trace Point Identifier
2	ThreadId: 0000004e	Thread Identifier (TID)
3	FunctionName: com.ibm.ws.sm.workspace.impl.WorkSpaceManagerImpl	Function name
4	SourceId: com.ibm.ws.sm.workspace.impl.WorkSpaceManagerImpl	Source Identifier
5	Category: AUDIT	Category
6, 7	ExtendedMessage: ...	Log message

Monitoring application logging using JMX notifications

Java developers can create programs to monitor application server logs using JMX notifications.

About this task

The most common log message listeners are written in Java, and connect to the deployment manager or an application server using SOAP. Use this topic to build a Java client that listens for log events.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Note: Be careful when adding listeners to servers with high logging volume as JMX notifications can slow down your server.

Procedure

1. Import the necessary packages. You will typically need the following import statements at the beginning of your Java program:

```
import javax.management.Notification;
import javax.management.NotificationListener;
import javax.management.ObjectName;
import javax.management.InstanceNotFoundException;
import javax.management.MalformedObjectNameException;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.exception.ConnectorException;
```

   Additionally, to handle the messages, and the types returned from the calls in subsequent steps you will need the following import statements.

```
import java.util.Iterator;
import java.util.Properties;
import java.util.Set;
import com.ibm.websphere.ras.RasMessage;
```

2. Create a Java class that implements the NotificationListener interface.
3. Implement the handleNotification method. The following example is a sample that writes the message text to the Java console:

```
public void handleNotification(Notification notification, Object handback) {
        RasMessage rasMessage = (RasMessage)notification.getUserData() ;
        System.out.println("Localized message: " + rasMessage.getLocalizedMessage(null));
}
```

4. Connect to the SOAP port of the server whose JMX MBeans you want to monitor. The following code creates a SOAP-connected AdminClient object with a specified host and a specified port:

```
        AdminClient adminClient = null ;
        String hostName = "someHostName";
        String soapPort = "8880";

        Properties connectProps = new Properties();
        connectProps.setProperty(AdminClient.CONNECTOR_TYPE, "SOAP");
        connectProps.setProperty(AdminClient.CONNECTOR_HOST, hostName);
        connectProps.setProperty(AdminClient.CONNECTOR_PORT, soapPort);

        try {
            adminClient = AdminClientFactory.createAdminClient(connectProps);
        } catch (ConnectorException e) {
            // error handling code
        }
```

5. Retrieve the MBean object name for the RasLoggingService MBean. The following code retrieves the RasLoggingService MBean object name:

```
        String queryString = "WebSphere:cell="+cellName+",node="+nodeName+",process="+serverName+",
type=RasLoggingService,*" ;
        Set<ObjectName> objectMBeans = null;
        try {
            ObjectName queryName = new ObjectName(queryString);
            objectMBeans = (Set<ObjectName>)adminClient.queryNames(queryName, null);
            if (objectMBeans.size() > 1) {
                // error handling code to deal with the case where we get more than one name returned.
            }
        } catch (MalformedObjectNameException e) {
            // error handling code
        } catch (ConnectorException e) {
            // error handling code
        }

        if (objectMBeans.isEmpty()) {
            // error handling code to deal with the case where the MBean is not found
        }

        Iterator<ObjectName> objectNames = objectMBeans.iterator() ;
        ObjectName objectName = objectNames.next() ;
```

6. Add the notification listener. This sample code adds a notification listener, waits for 60 seconds while it processes notifications, then removes the notification listener. A listener can stay connected as long as needed.

```
        try {
            adminClient.addNotificationListener(objectName, this, null, null);
            Thread.sleep(60 * 1000) ;
            adminClient.removeNotificationListener(objectName, this) ;
        } catch (InstanceNotFoundException e) {
            // error handling code
        } catch (ConnectorException e) {
            // error handling code
        } catch (Exception e) {
            // error handling code
        }
```

7. Add the necessary jar to your classpath. Add the admin client jar file to your classpath to be able to compile and run your code. The admin client jar file is in the <install_root>/runtimes directory.

Results

You have created a Java program that can listen to, and take actions as a result of log event notifications from an application server.

Setting up the error log

WebSphere Application Server for z/OS uses an error log to record error information when an unexpected condition or failure is detected within the product's own code. You can use the log stream to record activity and help diagnose problems.

About this task

Unexpected conditions or failures include:

- Assertion failures
- Unrecoverable error conditions
- Failures related to vital resources, such as memory
- Operating system exceptions
- Programming defects in WebSphere Application Server for z/OS code.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Procedure

- Because WebSphere Application Server for z/OS is predefined as a z/OS system logger application, you can use a log stream as the product's error log. By doing so, you can direct error information to a coupling facility log stream, which provides sysplex-wide error logging, or to a DASD-only log stream, which provides single system-only error logging.
- You can set up a common log stream for all WebSphere Application Server for z/OS servers, or individual log streams for each application server. Local z/OS client ORBs can also log data in log streams. The system logger APIs are unauthorized, but log stream resources can be protected using security products such as RACF®.
- You can use the WebSphere variable `ras_time_local` to control whether timestamps in the error log appear in local time (`ras_time_local=1`) or Greenwich Mean Time (GMT)(`ras_time_local=0`), which is the default.
- For additional information about z/OS log stream requirements, access the *z/OS MVS Setting up a Sysplex, SA22-7625* available on the z/OS Library web page

Viewing the service log

Service logs are logs written in a binary format. You cannot view a service log directly using a text editor. You should never directly edit the service log, as doing so will corrupt the log.

Before you begin

You can view a service log using 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.

About this task

Run the showlog script to view the contents of the service log as described in the following procedure.

Procedure

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

   Use the following format:

   ```
   showlog.sh {-start startDateTime [-end endDateTime] | -interval interval}
   [-format CBE-XML-1.0.1] [-encoding encoding] logStreamName
   [outputFilename]
   ```

   where:

   -start Specifies the start date and time, in yyyy-MM-ddTHH:mm:ss.SSSZ format. Milliseconds and time zone are optional.

   -end Specifies the end date and time, in yyyy-MM-ddTHH:mm:ss.SSSZ format. Milliseconds and time zone are optional.

-interval
Specifies the start date as the system date and time minus interval milliseconds, and end date as the system date and time. Valid values are integers greater than 0.

-format
Specifies the output format. Currently only CBE-XML-1.0.1 format is supported (this complies with the Common Base Event specification version 1.0.1). If no format is given, showlog outputs in a tabular format.

-encoding
Specifies the output file encoding, a character encoding supported by the local Java Virtual Machine .

logStreamName
Is a log file name.

outputFilename
Is optional. If no file name is given, the showlog script creates a default `showlog.out` filename, `outputFilename` is created in the current directory unless it is a fully qualified file name.

The formatted contents of the service log are always written to a file. There are parameters to showlog.sh which control content and encoding of the output. Enter `showlog.sh` without parameters for parameter usage information.

The showlog script can return informational messages containing service names, return codes, and reason codes. For more information about using the z/OS log stream, or to look up service names, return codes, and reason codes, refer to z/OS MVS Authorized Assembler Services Reference ENF-IXG(SA22-7610). Return and reason codes are listed for each service.

Refer to the topic “Authorization for System Logger Application Programs” in *z/OS MVS Assembler Services Guide (SA22-7605)* for advice on permitting access to the log stream.

4. Run the following showlog script with no parameters to display usage instructions.

   ```
   showlog.sh
   ```

5. Format and write the service log contents to a file.

   ```
   showlog service_log_filename output_filename
   ```

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

Example

Here are examples of showlog scripts on z/OS systems

- To write all records from the `WAS.ERROR.LOG` file since July 14, 2004 in log analyzer format into the `myoutput.log` file, use the following format:

  ```
  showlog.sh -start 2004-07-14T00:00:00 WAS.ERROR.LOG myoutput.log
  ```

- To write all records from `WAS.ERROR.LOG` file since July 14, 2004 in Common Base Event XML 1.0.1 format into `myoutput.log` file, use the following format:

  ```
  showlog.sh -start 2004-07-14T00:00:00 -format CBE-XML-1.0.1
  WAS.ERROR.LOG myoutput.log
  ```

- To write all records from `WAS.ERROR.LOG` file between July 14, 2004 and April 9, 2005 in Common Base Event XML 1.0.1 format into `myoutput.log` file, use the following format:

  ```
  showlog.sh -start 2004-07-14T00:00:00 -end 2005-04-09T00:00:00
  -format CBE-XML-1.0.1 WAS.ERROR.LOG myoutput.log
  ```

- To write all records from `WAS.ERROR.LOG` file since December 6, 2004 at 9pm Eastern standard time into `myoutput.log` file (the default output file), use the following format:

  ```
  showlog.sh -start 2004-12-06T21:00:00EST WAS.ERROR.LOG
  ```

Generating messages in Common Base Event format

Use the administrative console to enable writing of the logstream in Common Base Event format.

About this task

The z/OS logs can be stored in Common Base Event format. This enables the Showlog tool to read the data in the logstream. In turn, the showlog output can be read by the log and trace analyzer.

Procedure

1. Click **Servers** > **Server Types** > **WebSphere application servers** > **server1** > **Java and Process Management** > **Process Definition** > **Control** > **Java Virtual Machine** > **Custom Properties**.
2. Add a new custom property with `name="com.ibm.ws.logging.zOS.errorLog.format" and value "CBE-XML-1.0.1"`
3. Restart your application server for this setting to take effect.

Results

When this property is set to CBE-XML-1.0.1, the messages written to the error logstream are in binary Common Base Event format. You can then use the showlog script to view the binary Common Base Event records in the logstream.

Note:

> If you enable writing of the logstream in Common Base Event format, the error log is no longer viewable with the log browse utility. This action changes the format used to write to the logstream so that only the showlog tool can read it.

Logstream size considerations

You might need to modify the size of the logstream record size if the application server is attempting to write messages that are too large. If a message is too large, you will receive an error message that will be written to the job log.

If the logstream record size is too small for a message being written to it, you see a message similar to the following written to your job log:

```
TRAS0024I: Log entry is of size 5012 bytes which is too large to be added to
log stream which is configured for 4096 byte records.  Log entry will not be
logged to the log stream.
```

The original message is also written to the job log and can be viewed there.

To resolve this issue and ensure your messages fit into your logstream, change the MAXBUFSIZE of the error log logstream. The following code shows an example where the sample BBOERRLG job generated by the Profile Management Tool or the **zpmt** command is modified to set the MAXBUFSIZE to 8192:

```
//BBOERRLG  JOB (ACCTNO,ROOM),'USER10',CLASS=A,REGION=0M
//*
//*
//*
//BBORCLGS EXEC PGM=IXCMIAPU
//STDOUT DD  STDERR=*
//SYSIN    DD  *
    DATA TYPE(LOGR)
    DEFINE LOGSTREAM NAME(WAS.TY5.ERROR.LOG)
           DASDONLY(YES)
           HLQ(LOGGER)
           LS_SIZE(500)
           STG_SIZE(500)
           MAXBUFSIZE(8192)
           AUTODELETE(YES)
           RETPD(1)
           LS_DATACLAS(STANDARD)
```

Chapter 11. Working with trace

Use trace to obtain detailed information about running the WebSphere Application Server components, including application servers, clients, and other processes in the environment.

About this task

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.

You can configure trace settings with the administrative console, or you can configure tracing from the MVS console using the modify command.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Procedure

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.

Results

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

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

Enabling trace on client and stand-alone applications

When stand-alone client applications (such as Java applications which access enterprise beans hosted in WebSphere Application Server) have problems interacting with WebSphere Application Server, it might 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.

About this task

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.

You can also configure tracing from the MVS console using the modify command.

Procedure

1. To enable trace for the WebSphere Application Server classes in a client application, add the system properties shown in the following example to the startup script or command of the client application. The location of the output and the classes and detail included in the trace follow the same rules as for adding trace to WebSphere Application Servers. For example, trace the stand-alone client application program named `com.ibm.sample.MyClientProgram`, enter the following command:

   ```
   java -DtraceSettingsFile=MyTraceSettings.properties
   -Djava.util.logging.manager=com.ibm.ws.bootstrap.WsLogManager
   -Djava.util.logging.configureByServer=true com.ibm.samples.MyClientProgram
   ```

   The file identified by *file name* must be a properties file placed in the class path of the application client or stand-alone process. You must 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.

   You cannot use the **-DtraceSettingsFile=TraceSettings.properties** property to enable tracing of the ORB component for thin clients. ORB tracing output for thin clients can be directed by setting **com.ibm.CORBA.Debug.Output = debugOutputFilename** parameter in the command line.

   The java.util.logging.manager and java.util.logging.configureByServer system properties configure Java logging to use a WebSphere Application Server-specific LogManager class and to use the configuration from the file specified by the traceSettingsFile property. The default Java Logging properties file, located in the Java SE Runtime Environment 6 (JRE6), will not be applied.
2. 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.

Results

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.

Enabling trace at server startup

Use the administrative console to enable tracing at a server's startup. You can use trace to assist you in monitoring system performance and diagnosing problems.

About this task

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.

You can also configure tracing from the MVS console using the modify command.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Procedure

1. Start the administrative console.
2. Click **Servers > Application Servers >** *server_name* **> Troubleshooting > Diagnostic Trace Service**.
3. Click **Configuration**.
4. Select whether to direct trace output to either a file or an in-memory circular buffer.

   Note: Different components can produce different amounts of trace output per entry. Naming and security tracing, for example, produces a much higher trace output than web container tracing. Consider the type of data being collected when you configure your memory allocation and output settings.
5. 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.
6. 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.
7. Select the desired format for the generated trace.
8. Save the changed configuration.
9. To enter a trace string to set the trace specification to the desired state:
   a. Click **Troubleshooting > Logs and trace** 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 the page on log level settings. 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**.
10. Allow enough time for the nodes to synchronize, and then start the server.

Enabling trace on a running server

Use the administrative console to enable tracing on a running server. You can use trace to assist you in monitoring system performance and diagnosing problems.

About this task

You can modify the trace service state that determines which components are being actively traced for a running server by using the following procedure.

You can also configure tracing from the MVS console using the modify command.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Procedure

1. Start the administrative console.
2. Select the **Runtime** tab.
3. Select the **Save runtime changes to configuration as well** check box if you want to write your changes back to the server configuration.
4. Change the existing trace state by changing the trace specification to the desired state. Refer to the following topics for specific information on how to change your trace specifications:
   - "Diagnostic trace service settings" describes how you can examine the current diagnostic trace settings and to change where the trace is written (memory buffer or file). If the trace is to be recorded in a memory buffer, you can specify the file to which the memory buffer is to be dumped. You can view the file for trace information.
   - "Log and trace settings" describes how you can view and configure logging and trace settings for the server.
   - "Managing the application server trace service" describes how 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. You can select where the trace is written (memory buffer or file).

   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.
5. Configure the trace output if a change from the existing one is desired.
6. Click **Apply**.

Select a server to configure logging and tracing

Use this page to select the server for which you want to configure logging and trace settings.

This page lists application servers in the cell and the nodes holding the application servers. The status indicates whether a server is running, stopped, or encountering problems.

If you are using the WebSphere Application Server, Network Deployment product, this page also shows the status of the application servers.

When you select an application server, a page 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

Specifies the logical name of the server.

Node

Specifies the name of the node for the application server.

Host name

Specifies the name of the host for the application server.

Version
Specifies the version for the application server.

Type
Specifies the type of application server.

Status
Indicates whether the application server is started or stopped. (WebSphere Application Server, Network Deployment only)

Note that if the status is *Unavailable*, the node agent is not running in that node, and you must restart the node agent before you can start the server.

Log and trace settings

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

Note: You can only access this page when the server is configured to use basic log and trace mode.

To view this administrative console page, click **Troubleshooting** > **Logs and trace** > ***server_name*** .

Note: You can configure tracing from the MVS console using the modify command.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

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

NCSA access and HTTP error logging
The NCSA access and HTTP error logging page enables you to configure the log settings for your HTTP server.

Change log and trace mode
Specifies to switch between HPEL mode or Basic mode logging.

Note: Switching the server to HPEL log and trace mode requires a server restart.

Setting up component trace (CTRACE)

WebSphere Application Server for z/OS uses z/OS component trace (CTRACE) facilities to manage the collection and storage of trace data. CTRACE data is written to address space buffers in private (pageable) storage, which can be formatted using IPCS if a dump of the address space is taken. CTRACE data can also be written to trace data sets on disk or tape using an external writer.

Before you begin

Although CTRACE data is primarily output for use by IBM service personnel, using CTRACE capabilities at your installation allows you to have additional trace data available when a problem first occurs. Because CTRACE efficiently uses system resources, you can collect valuable trace data with minimal impact on performance. For detailed information about the CTRACE facilities, see *z/OS MVS Diagnosis: Tools and Service Aids, GA22-7589*.

About this task

If you choose to write CTRACE data to trace data sets, you must create an external writer. You can set up separate trace data sets for each cell or for each WebSphere Application Server for z/OS release, or you can use a single trace data set for all WebSphere Application Server activity on a particular z/OS system.

Procedure

To implement a CTRACE data trace, read the following articles on preparing and starting CTRACE in your application server:

- "Preparing CTRACE controls and resources"
- "Starting CTRACE as part of WebSphere Application Server for z/OS initialization" on page 164
- "Starting CTRACE while WebSphere Application Server for z/OS servers are active" on page 164
- "CTRACE to collect trace data for Java server applications" on page 165

Results

After you read articles you will be able to implement CTRACE data tracing in your applications.

Preparing CTRACE controls and resources

You must prepare CTRACE controls and resources before using it for trace data.

Before you begin

Before you start component trace (CTRACE) activity for WebSphere Application Server for z/OS servers, you need to make some decisions about CTRACE controls and resources as well as create an external writer if one is needed for trace data recording.

About this task

Perform the following steps to prepare CTRACE controls and resources:

Procedure

1. Decide whether to write CTRACE data to trace data sets (recommended) or keep CTRACE data in memory buffers only.
2. If you wish to use trace data sets, perform these steps.
   a. Decide whether to create one trace data set for all WebSphere Application Server activity on a single z/OS system or separate trace data sets for each cell or WebSphere Application Server release.

      Trace data sets cannot be shared between z/OS systems; each system should have its own trace data sets.
   b. Choose names for the trace data sets.

      To simplify external writer setup, include the z/OS system name in the data set name.

      Recommendation: For a single trace data set for all WebSphere Application Server activity, use something similar to SYS1.*sysname*.WAS390.CTRACE.

c. Allocate a trace data set on each z/OS system.

Note:

Do not specify DCB parameters RECFM, LRECL, or BLKSIZE; the external writer will allocate them with record format VB and a system-optimal blocksize and logical record length. For trace data sets on disk, you should use a minimum of 10 cylinders (3390). Secondary extents are ignored unless the NOWRAP option is specified when the external writer is started. For example:

```
// EXEC PGM=IEFBR14
//TRACE DD DSN=SYS1.MVSS14.WAS390.CTRACE,UNIT=3390,VOL=SER=HPK19A,
//   SPACE=(CYL,(20,0)),DISP=(NEW,CATLG),DCB=DSORG=PS
```

d. Choose a job name for the external writer.

Recommendation: Use BBOWTR if the same trace data set is to be used for all WebSphere Application Server activity on each z/OS system.

e. Create the external writer cataloged procedure.

1) Copy member BBOWTR from the WebSphere Application Server for z/OS product data set SBBOJCL to SYS1.PROCLIB or another procedure library defined to the master scheduler.
2) Rename the procedure to the external writer job name that you chose.
3) Customize the cataloged procedure by providing your trace data set name where indicated.
4) If the cataloged procedure will be shared among several z/OS systems, make sure that the trace data set DD statements point to the appropriate trace data sets on each system.

f. Choose a system user ID under which the external writer started task will run.

This user ID must have read/write access to the trace data sets; you may wish to use an existing started task user ID such as the default started task user ID for your system. Use the following RACF command or equivalent to cause the external writer cataloged procedure to run under the started task user ID:

```
RDEFINE STARTED external_writer_procname.* STDATA(USER(system_user_ID)) TRACE(YES)
```

The external writer started task should run with at least as high a dispatching priority as the WebSphere Application Server address spaces that will use it for tracing.

g. Start the external writer to verify that the steps above were performed correctly.

Enter the following MVS console command:

```
TRACE CT,WTRSTART=external_writer_procname
```

3. Create a CTRACE parmlib member.

a. Copy member BBOCTI00 from the WebSphere Application Server for z/OS product data set SBBOJCL to a data set in your system parmlib concatenation.
b. Rename the parmlib member to CTIBBO*xx*, where *xx* is a two-character suffix to be associated with the external writer.

This is the value that will be specified during WebSphere Application Server for z/OS customization.

c. Customize the CTIBBO*xx* parmlib member to indicate whether trace data sets and an external writer are to be used and, if so, whether the external writer should be started automatically when WebSphere Application Server activates the CTRACE.

4. If you plan to use separate trace data sets for different WebSphere Application Server cells or releases, repeat all of these steps while choosing a new external writer name and parmlib member suffix for each.

Results

CTRACE for WebSphere Application Server is now set up. Use the parmlib member suffix to associate a particular WebSphere Application Server for z/OS cell with the CTRACE options that you have chosen.

Starting CTRACE as part of WebSphere Application Server for z/OS initialization

You can start CTRACE as part of the initialization process for a WebSphere Application Server for a z/OS cell using this information.

Before you begin

Make sure that you have properly prepared CTRACE controls and resources as described in “Preparing CTRACE controls and resources” on page 162.

About this task

Perform the following steps to start CTRACE as part of the initialization process for a WebSphere Application Server for z/OS cell:

Procedure

1. Start the CTRACE external writer.
   - If you want the external writer to write records to the trace data set until the existing extents are full then overwrite the oldest records, use the following MVS console command:

     ```
     TRACE CT,WTRSTART=procname
     ```

     where *procname* is the name of the cataloged procedure for the CTRACE writer.
   - If you want the external writer to fill all primary and secondary extents then terminate, add the NOWRAP option as in the following example:

     ```
     TRACE CT,WTRSTART=procname,NOWRAP
     ```
   - If the CTIBBO*xx* member for the cell contains a WTRSTART parameter, then no command is necessary. If the external writer is not started, WebSphere Application Server will start it automatically.
2. Start the WebSphere Application Server for z/OS application server using the generated instructions.
3. When you need to collect trace data for analysis, perform these steps.
   a. Disconnect WebSphere Application Server for z/OS from CTRACE.
      1) Use the following operator command:

         ```
         TRACE CT,ON,COMP=cell_short_name
         ```
      2) You will be prompted for additional options. Enter the following reply:

         ```
         REPLY x,WTR=DISCONNECT,END
         ```
   b. Stop the CTRACE external writer.

      Use the following operator command:

      ```
      TRACE CT,WTRSTOP=procname
      ```

      where *procname* is the name of the cataloged procedure for the CTRACE writer.

Starting CTRACE while WebSphere Application Server for z/OS servers are active

Use this page to start CTRACE when a WebSphere Application Server for z/OS server already is active.

Before you begin

Make sure that you have properly prepared CTRACE controls and resources as described in “Preparing CTRACE controls and resources” on page 162.

About this task

If you start a WebSphere Application Server for z/OS server before starting the CTRACE writer for WebSphere, the server still gathers data in its trace buffers. This trace data is not available for use unless you follow this procedure or until a dump of the server address space is taken.

Perform the following steps to start CTRACE when a WebSphere Application Server for z/OS server already is active:

Procedure

1. Perform the following tasks.
   a. Start the CTRACE external writer.
      Use the following operator command:
      ```
      TRACE CT,WTRSTART=procname
      ```
      where *procname* is the name of the cataloged procedure for the CTRACE writer.
   b. If necessary, connect WebSphere Application Server for z/OS to a CTRACE writer other than the one specified in the CTIBBO *xx* parmlib member.
      1) Use this operator command:
         ```
         TRACE CT,ON,COMP=cell_short_name
         ```
      2) You will be prompted for additional options. Enter the following reply:
         ```
         REPLY x,WTR=procname,END
         ```
         where *procname* is the name of the cataloged procedure for the CTRACE writer.

   The CTRACE external writer begins writing the server's trace data to the location specified through the WebSphere variable *ras_trace_outputLocation*.
2. When you need to collect trace data for analysis, perform these steps.
   a. Disconnect WebSphere Application Server for z/OS from CTRACE.
      1) Use the following operator command:
         ```
         TRACE CT,ON,COMP=cell_short_name
         ```
      2) You will be prompted for additional options. Enter the following reply:
         ```
         REPLY x,WTR=DISCONNECT,END
         ```
   b. Stop the CTRACE external writer.
      Use the following operator command:
      ```
      TRACE CT,WTRSTOP=procname
      ```
      where *procname* is the name of the cataloged procedure for the CTRACE writer.

CTRACE to collect trace data for Java server applications

Applications that run in WebSphere Application Server for z/OS can use JRas to provide tracing support that is consistent with WebSphere tracing.

Instrumented applications use the JRas interfaces and classes for logging and tracing; trace data is written to the same component trace data set as the internal traces issued by the WebSphere Application Server for z/OS runtime. So you can gather application trace data in the same locations, and use the same commands to start and stop CTRACE for these JRas applications as you do for WebSphere Application Server for z/OS server in which the applications are running.

Chapter 12. Troubleshooting class loaders

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

Before you begin

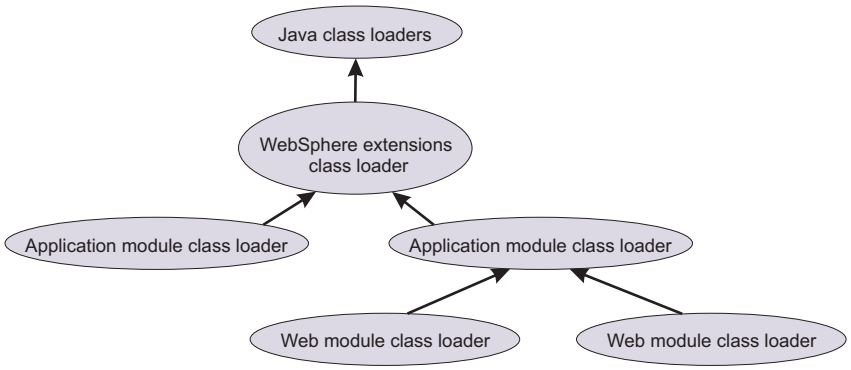
This topic assumes that you have installed an application on a server supported by the product and you want to examine class loaders used by the application or its modules. The modules can be web modules (`.war` files) or enterprise bean (EJB) modules (`.jar` files). The class loader viewer enables you to examine class loaders in a runtime environment.

This topic also assumes that you have enabled the class loader viewer service. Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name*** > **Class loader viewer service**, enable the service and restart the server.

About this task

The runtime environment of WebSphere Application Server uses the following class loaders to find and load new classes for an application in the following order:

1. The bootstrap, extensions, and CLASSPATH class loaders created by the Java virtual machine
2. A WebSphere extensions class loader
3. One or more application module class loaders that load elements of enterprise applications running in the server
4. Zero or more web module class loaders



Each class loader is a child of the previous class loader. That is, the application module class loaders are children of the WebSphere extensions class loader, which is a child of the CLASSPATH Java class loader. Whenever a class needs to be loaded, the class loader usually delegates the request to its parent class loader. If none of the parent class loaders can find the class, the original class loader attempts to load the class. Requests can only go to a parent class loader; they cannot go to a child class loader. After a class is loaded by a class loader, any new classes that it tries to load reuse the same class loader or go up the precedence list until the class is found.

If the class loaders that load the artifacts of an application are not configured properly, the Java virtual machine (JVM) might throw a class loading exception when starting or running that application. “Class

loading exceptions" on page 169 describes the types of exceptions caused by improperly configured class loaders and suggests ways to use the class loader viewer to correct configurations of class loaders. The types of exceptions include:
- ClassCastException
- ClassNotFoundException
- NoClassDefFoundException
- UnsatisfiedLinkError

Use the class loader viewer to examine class loaders and correct problems with application or class loader configurations.

Procedure

- Examine a tree view that lists all installed applications and their modules. The modules can be web modules (`.war` files) or EJB modules (`.jar` files).

  Click **Troubleshooting** > **Class loader viewer** to access the Enterprise applications topology page.
- Examine the class loader delegation hierarchy.

  On the Enterprise applications topology page, select a module to access the Class loader viewer page. The page lists the class loaders visible to web and EJB modules in an installed enterprise application. This page helps you to determine which class loaders loaded files of a module and to diagnose problems with class loaders.

  The delegation hierarchy is determined by the class loader delegation mode, or *class loader order*, specified for an application or web module. The value can be either `Classes loaded with parent class loader first` or `Classes loaded with local class loader first (parent last)`. Refer to the Configure class loaders step for more information.
- Export information on class loaders.
  1. On the Class loader viewer page, click **Export**.
  2. Select to open a browser or editor on the class loader information or to save the information to disk in XML format.
  3. Click **OK**, and specify any additional information requested by the system.
- Display information about class loaders visible to the module in an HTML table format.

  On the Class loader viewer page, click **Table View**. The Table View page displays the following information:

Table 35. Table View page. Information available on class loader attributes.

Class loader attribute	Description
Delegation	Indicates whether the class loader delegates the loading of the module to its parent class loader. A value of `true` implies that the class loader of the parent application is being used (`Classes loaded with parent class loader first`). A value of `false` implies that the module class loader is being used (`Classes loaded with local class loader first (parent last)`). Refer to the Configure class loaders step for more information.
Classpath	Lists the paths over which the class loader searches for classes and resources.
Classes	Lists the names of classes loaded in the JVM by this class loader.

  The **Table View** option does not return a value when out-of-memory errors are generated. The out-of-memory errors might be related to a memory leak. To examine information about class loaders in a table, resolve the out-of-memory problem, and then click **Table View** again.
- Search class loaders.

  On the Class loader viewer page, click **Search** to access the Search page, on which you can search class loaders for the following:
  - Specific strings
  - Specific `.jar` files
  - The names of files in a specific directory

– The names of files loaded by a specific class loader

The search is case-sensitive. “Class loading exceptions” describes several uses of the Search page.

- Configure class loaders. You can configure class loaders for the following:
  – All applications installed on a specific server.
  – A specific application
  – A specific web module

  Note: For detailed information about server, application, and web class loaders, see the chapter on class loading in the *Developing and deploying applications* PDF book.

  Class loader configuration determines which class loader loads the classes and resource files for an application or web module. Application and WAR module class loader configuration settings include **Class loader order** and **WAR class loader policy**.

  A **Class loader order** value can be either `Classes loaded with parent class loader first` or `Classes loaded with local class loader first (parent last)`. The default is `Classes loaded with parent class loader first`. A class loader with the `Classes loaded with parent class loader first` mode delegates loading a class or resource to its immediate parent class loader before searching its classpath.

  When troubleshooting class loading problems, you might need to override classes visible to a parent class loader. To override such classes with those specific to an application, set the **Class loader order** to `Classes loaded with local class loader first (parent last)` on the class loader that contains the application classes on its classpath. An application can override classes visible to a parent class loader, but doing so can result in a ClassCastException or UnsatisfiedLinkError if there is a mixed use of overridden classes and non-overridden classes.

  For example, under default class loader policies, a web module has its own Web module (WAR) class loader to load its artifacts, which are typically in the `WEB-INF/classes` and `WEB-INF/lib` directories. An application module class loader is the immediate parent of this WAR class loader. To ensure that the web module class loader searches these paths for a particular class or resource first, before delegating the load operation to the application module class loader, set the **Class loader order** of the web module to `Classes loaded with local class loader first (parent last)`.

  Class loader policies determine the structure of the application and WAR module class loaders. Under the default policies, every running application EAR has its own application module class loader, and every web module has its own WAR module class loader. The default policies ensure Java EE compliance regarding visibility and isolation among application artifacts. Changing the default policies is not suggested when troubleshooting class loading problems.

What to do next

If you continue to have class loader problems, refer to “Class loading exceptions” and to the class loading chapter of the *Developing and deploying applications* PDF book.

Class loading exceptions

What kind of class-loading error do you see when you develop an application or start an installed application?

ClassCastException

A class cast exception results when the following conditions exist and can be corrected by the following actions:

- The type of the source object is not an instance of the target class (type).

- The class loader that loaded the source object (class) is different from the class loader that loaded the target class.
- The application fails to perform or improperly performs a narrow operation.

The type of the source object is not an instance of the target class (type).

This is the typical class cast exception. You can diagnose whether the source object of a cast statement is not an instance of the target class (type) by examining the class signature of the source object class, then verifying that it does not contain the target class in its ancestry and the source object class is different than the target class. You can obtain class information by inserting a simple print statement in your code. For example:

```
System.out.println( source.getClass().getName() + ":" + target.getClass().getName() );
```

Or use a **javap** command. For example:

```
javap java.util.HashMap
Compiled from "HashMap.java"
public class java.util.HashMap extends java.util.AbstractMap
             implements java.util.Map,java.lang.Cloneable,java.io.Serializable {
```

The class loader that loaded the source object (class) is different from the class loader that loaded the target class.

Assuming that the type of the source object is an instance of the target class, a class cast exception occurs when the class loader that loaded the source object's class is different that the class loader that loaded the target class. This condition might occur when the target class is visible on the classpaths of more than one class loader in the WebSphere Application Server runtime environment. To correct this problem, use the Search and Search by class name console pages used to diagnose problems with class loaders:

1. Click **Troubleshooting** > **Class loader viewer** > ***module_name*** > **Search** to access the Search page.
2. For **Search type**, select **Class/Package**.
3. For **Search terms**, type the name of the class that is loaded by two class loaders.
4. Click **OK**. The Search by class name page is displayed, listing all class loaders that load the class.

   If there is more than one class loader listed, then the target class was loaded by more than one class loader. Because the source object is an instance of the target class, the class loader that loaded the source object class is different from the class loader that loaded the target class.
5. Return to the Class loader viewer page and examine the classpath to determine why two different class loaders load the class.
6. Correct your code so that the class is visible only to the appropriate class loader.

The application fails to perform or improperly performs a narrow operation.

A class cast exception can occur because, when the application is resolving a remote enterprise bean (EJB) object, the application code does not perform a narrow operation as required. The application must perform a narrow operation after looking up a remote object. Examine the application and determine whether it looks up a remote object and, if so, the result of the lookup is submitted to a narrow method.

The narrow method must be invoked according to the EJB 2.0 programming model. In particular, the target class submitted to the narrow method must be the exact, most derived interface of the EJB. This also causes a class cast exception in the WebSphere Application Server runtime environment. Examine the application and determine whether the target class submitted to the narrow method is a super-interface of the EJB that is specified, not the exact EJB type; if so, modify the application to invoke narrow with the exact EJB interface.

Lastly, if a class cast exception occurs during a narrow operation, verify that the narrow method is being applied to the result of a remote EJB lookup, not to a local enterprise bean. A narrow is not used for local lookups. Examine the application or module deployment descriptor to ensure that the object being narrowed is not a local object.

ClassNotFoundException

A class not found exception results when the following conditions exist and can be corrected by the following actions:

- The class is not visible on the logical classpath of the context class loader.
- The application incorrectly uses a class loader API.
- A dependent class is not visible.

The class is not visible on the logical classpath of the context class loader.

The class not found is not in the logical class path of the class loader associated with the current thread. The logical classpath is the accumulation of all classpaths searched when a load operation is invoked on a class loader. To correct this problem, use the Search page to search by class name and by Java archive (JAR) name:

1. Click **Troubleshooting** > **Class loader viewer** > ***module_name*** > **Search** to access the Search page.
2. For **Search type**, select **Class/Package**.
3. For **Search terms**, type the name of the class that is not found.
4. Click **OK**. The Search by class name page is displayed, listing all class loaders that load the class.
5. Examine the page to see if the class exists in the list.
6. If the class is not in the list, return to the Search page. For **Search terms**, type the name of the `.jar` file for the class; for **Search type**, select **JAR/Directory**.
7. Click **OK**. The Search by Path page is displayed, listing all directories that hold the JAR file.

If the JAR file is not in the list, the class likely is not in the logical class path, not readable or an alternate class is already loaded. Move the class to a location that enables it to be loaded.

The application incorrectly uses a class loader API.

An application can obtain an instance of a class loader and call either the loadClass method on that class loader, or it can call Class.forName(*class_name*, *initialize*, *class_loader*) with that class loader. The application may be incorrectly using the class loader application programming interface (API). For example, the class name is incorrect, the class is not visible on the logical classpath of that class loader, or the wrong class loader was engaged.

To correct this problem, determine whether the class exists and whether the application is properly using the class loader API. Follow the steps in The class is not visible on the logical classpath of the context class loader to determine whether the class is loaded. If the class has not been loaded, attempt to correct the application and see if the class loads. If the class is in the class path with proper permission and is not being overridden by another factory class, examine the API used to load the class.

1. Click **Troubleshooting** > **Class loader viewer** > ***module_name*** > **Search** to access the class loader Search page.
2. For **Search type**, select **Class/Package**.
3. For **Search terms**, type the name of the class.
4. Click **OK**. The Search by class name page is displayed, listing all class loaders that load the class.
5. Examine the page to see if the class exists in the list.
6. If the class is in the list and a ClassNotFound exception was thrown, then the `.jar` file or class is not in the correct context or a wrong API call in the current context was used.

   If the class is not in the list, return to the Search page and do the following:
   a. Search for the class that generated the exception; that is, the class calling Class.forName.
   b. See which class loader loads the class.
   c. Determine whether the class loader has access or can load the class not found by evaluating the class path of the class loader.

A dependent class is not visible.

When a class loader *clsldr* loads a class *cls*, the Java virtual machine (JVM) invokes *clsldr* to load the classes on which *cls* depends. Dependent classes must be visible on the logical classpath of *clsldr*, otherwise an exception occurs. This condition typically occurs when users make WebSphere Application Server classes visible to the JVM, or make application classes visible to the JVM or to the WebSphere extensions class loader. For example:

- Class A depends on Class B.
- Class A is visible to the WebSphere extensions class loader.
- Class B is visible on the local classpath of a WAR module class loader, not the WebSphere extensions class loader classpath.

When the JVM loads class A using the WebSphere extensions class loader, it then attempts to load Class B using the same class loader and ultimately creates a class not found exception.

To correct this problem:

1. Make the application-specific classes visible to the appropriate application class loader.
2. Search for the class not found (Class B).
3. If Class B is in the proper location, search for the class that loads the dependent class (Class A) in the Class loader viewer.
4. If the class is loaded and a ClassNotFound exception was thrown, then the `.jar` file or class is not in proper context or the wrong API call in the current context was used.

   If no class was found, do the following:
   a. Search for the class that generated the exception; that is, the class calling Class.forName.
   b. See which class loader loads the class.
   c. Determine whether the class loader has access or can load the class not found by evaluating the class path of the class loader.
5. Ensure that the caller class (Class B) is visible to the JVM or WebSphere extensions class loader.

NoClassDefFoundException

A no class definition found exception results when the following conditions exist and can be corrected by the following actions:

The class is not in the logical class path.

Refer to “ClassNotFoundException” on page 171 for information.

The class cannot load.

There are various reasons for a class not loading. The reasons include: failure to load the dependent class, the dependent class has a bad format, or the version number of a class.

UnsatisfiedLinkError

A linkage error results when the following conditions exist and can be corrected by the following actions:

- A user action caused the error.
- The native library is already loaded.
- A dependent native library was used.

A user action caused the error.

Several user actions can result in a linkage error:

A library extension name is incorrect for the platform.

System.loadLibrary is passed an incorrect parameter.

The library is not visible.

As a best practice, use the JVM class loader to find or load native libraries. WebSphere Application Server prints the Java library path (`java.library.path`) when starting up. If the JVM class loader is intended to load the library, verify that the path containing the native

library file is in the Java library path. If not, append the path to the platform-specific native library environment variable or to the `java.library.path` system property of the server process definition.

In general, the Java virtual machine invokes findLibrary() on the class loader *xxx* that loads the class that calls System.loadLibrary(). If *xxx*.findLibrary() fails, the Java virtual machine attempts to find the library using the JVM class loader, which searches the JVM library path. If the library cannot be found, the Java virtual machine creates an UnsatisfiedLinkError exception.

Thus, if a WebSphere class loader is intended to find a native library `myNativeLib`, the library must be visible on the `nativelibpath` of the class loader that loads the class that calls System.loadLibrary(`myNativeLib`). This practice is necessary or desirable in the following situation:

- Native libraries for data source providers must be visible on the `nativelibpath` of the WebSphere extensions class loader. In this case, add the path containing the native library to the **Native library path** setting of the data source provider configuration.
- Shared libraries have a **Native library path** in their configuration. Because shared libraries enable the versioning of application-specific libraries, consider specifying the paths to any native libraries used by the shared library code in the shared library configuration.

Ensure that the correct WebSphere class loader loads the class that calls System.loadLibrary() and that the native library is visible on the **Native library path** setting.

The native library is already loaded.

This condition can result from either of the following errors:

User error

Check for multiple calls to System.loadLibrary and remove any extraneous calls.

Error when an application restarts

The JVM has a restriction that only one class loader can load a native library at a time. An error results when an application restarts before the garbage collector cleans up the class loader from the stopped application. When the class that loads the native library moves, all of the classes that depend on that native library and their dependencies also must move.

To correct this condition, move the loading of the native library to a class loader that does not reload:

1. Locate all application classes that load native libraries or have native methods.
2. Identify any dependent classes for the classes in step 1, such as logging packages.
3. Create a server-associated shared library or an isolated shared library.
4. Move the JAR files loaded for classes in steps 1 and 2 from the application to the shared library created in step 3.
5. Save your changes.
6. Redeploy the application and rerun the scenario.

For more information about invoking, creating, and managing shared libraries, read “Managing shared libraries” in the *Administering applications and their environment* PDF book.

Classes within server-scoped libraries are loaded once for each server lifecycle, ensuring that the native library required by the application is loaded once for each Java virtual machine, regardless of the application's life cycle.

A dependent native library was used.

Dependent native libraries must be found or loaded by the JVM class loader. That is, if a native library *NL* is dependent on another native library, *DNL*, the JVM class loader must find DNL on the Java library path. This is because the JVM runs native code when loading NL; when it encounters

the dependency on DNL, the JVM native code can call only to the JVM class loader to resolve the dependency. A WebSphere class loader cannot load a dependent native library.

Modify the platform-specific environment variable defining the Java library path (LIBPATH) to include the path containing the unresolved native library.

osgiCfgInit script

The Equinox OSGi framework is used to manage class loading and relationships between server component bundles. In some cases, the cached bundle data, which is maintained on a per-profile basis and has a separate cache at the WAS_HOME level for installation-wide processes, can become out of sync with the actual binaries on the server. You can use the osgiCfgInit script to clear and recreate the OSGi cache.

You should run the osgiCfgInit script on the command line from the `WAS_HOME/bin` or *user_install_root*`/bin` directory. The behavior of the script depends on the directory from which you run the script. If you run the script from a profile-level bin directory, the script clears the OSGi cache for all servers within that profile. If you run the script from the `WAS_HOME/bin` directory, the script clears the OSGi cache for all servers within the default profile.

Note: Before you tun the osgiCfgInit script, stop the server on which the script will be run. If you run this script on a server that is active, the server might have problems trying to read or update the cache after the script is finished.

Syntax

The syntax for this script is as follows:

```
osgiCfgInit.sh [options]
```

Options

The following options are available for the osgiCfgInit script:

-all
The script clears the caches of all servers in the installation, as well as the WAS_HOME cache.

-washome
The script clears the cache at the WAS_HOME level.

Usage scenario

To clear the cache, execute the command as follows:

```
./osgiCfgInit.sh [-all|-washome]
```

If the script completes successfully, the message "OSGi cache successfully cleaned for *location*." displays on the command line.

Class loader viewer service settings

Use this page to configure the server to start the class loader viewer service when the server starts. The Class Loader Viewer helps you diagnose problems with class loaders.

To view this administrative console page, click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name*** > **Class loader viewer service**.

Class loaders find and load class files. For a deployed application to run properly, the class loaders that affect the application and its modules must be configured so that the application can find the files and

resources that it needs. Diagnosing problems with class loaders can be complicated and time-consuming. To diagnose and fix the problems more quickly, enable the class loader viewer service on this page and then use the console Class loader viewer to examine class loaders and the classes loaded by each class loader. Click **Troubleshooting** > **Class loader viewer** to access the Class loader viewer in the console.

Enable service at server startup

Specifies whether or not the server attempts to start the class loader viewer service when the server starts.

The default is not to start the class loader viewer service.

Enterprise application topology

Use this page to see where modules reside in a topology of enterprise applications. Knowing where a module resides helps you to determine which class loader loaded a module and to diagnose problems with class loaders.

To view this administrative console page, click **Troubleshooting** > **Class loader viewer**. This page lists all installed applications and their modules in a tree view. The modules can be web modules (`.war` files) or enterprise bean (EJB) modules (`.jar` files).

When deploying an application to a server or starting an application, you might encounter problems related to class loaders. Use the console pages accessed from this page to troubleshoot errors such as the following:

- ClassCastException
- ClassNotFoundException
- NoClassDefFoundException
- UnsatisfiedLinkError

You can use the Class loader viewer console pages without having to restart or manipulate the application.

Enterprise applications topology

Displays a tree hierarchy of applications installed on a server and lists the module files in the class paths of the applications.

Expand the hierarchy for an application to see what web modules (`.war` files) and EJB modules (`.jar` files) are in the application class path.

Click on a module name to examine the class loaders of the module.

Class loader viewer settings

Use this page to examine the class loaders visible to a web module (`.war` file) or enterprise bean (`.ejb` file) in an installed enterprise application. This page helps you to determine which class loaders loaded files of a module and to diagnose problems with class loaders.

To view this administrative console page, click **Troubleshooting** > **Class loader viewer** > ***module_name***.

The module is currently running on all nodes and servers listed.

To learn more about classes used by the module and their class loaders, click a button:

Table 36. Class loader viewer buttons. Click a button to access information about classes.

Button	Resulting action
Export	Opens a dialog that enables you to view or save the class loader information on this page in an XML file.
Table View	Displays the Table view page, which provides information about class loaders visible to the module in an HTML table format for each class loader. Such information includes: **Delegation** Whether the class loader delegates a load operation to its immediate parent before searching its local classpath for a class or resource **Classpath** The local classpath, which includes the paths over which the class loader searches for classes and resources, excluding the classpaths of any parent class loaders. **Classes** The names of classes loaded by the class loader
Search	Displays the Search page, on which you can search class loaders for the following: • Specific strings • Specific `.jar` files • The names of files in a specific directory • The names of files loaded by a specific class loader

Class Loader

Displays a hierarchy of class loaders that affect the loading of classes used by the web or EJB module. The **Hierarchy** tab displays the class loaders in a tree hierarchy. The **Search Order** tabs lists the class loaders in the order in which the runtime environment uses them to find and load classes.

Expand a hierarchy of class loaders to view the following:

- Class loader names
- Arrows that point upwards beside class loader names, indicating that requests can go to a parent class loader only and not go to a child class loader
- The names of classes that are loaded by a class loader
- The paths of property files and `.jar` files used by the classes

The following class loaders might be in a hierarchy:

Table 37. Class loader name descriptions. Class loaders that might be in the hierarchy of class loaders.

Class loader name	Description
JDK Extension Loader	The JDK extensions class loader is a composite class loader that is comprised of the Java virtual machine (JVM) bootstrap class loader, the JVM extensions class loader and the JVM system class loader, which load the core SDK classes and resources as well as classes and resources visible on the JVM classpath.
WAS Extension Class Loader	The WAS Extension Class Loader loads the WebSphere Application Server classes, stand-alone resource classes, custom service classes, and custom registry classes. At bootstrap, this class loader uses the `ws.ext.dirs` system property to determine the path that is used to load classes. Each directory in the `ws.ext.dirs` class path and every `.jar` file or compressed `.zip` file in these directories is added to the class path used by this class loader.

Table 37. Class loader name descriptions (continued). Class loaders that might be in the hierarchy of class loaders.

Class loader name	Description
WAS Compound Class Loader	The WAS Compound Class Loaders load classes and resources of enterprise archive (EAR) modules, web application archive (WAR) modules, and server-associated shared libraries. Under default class loader policies, an instance of a WAS Compound Class Loader exists for each running EAR and WAR module and for each class loader defined in the server configuration.

Click on **Classes** to view a list of classes loaded by a class loader.

The class loader viewer service must be enabled to view the list of classes.

Search settings

Use this page to search for information about class loaders visible to a web module (`.war` file) or enterprise bean (`.ejb` file) in an installed enterprise application. This page helps you diagnose problems with class loaders.

To view this administrative console page, click **Troubleshooting** > **Class loader viewer** > ***module_name*** > **Search**.

On the Search page, you can search class loaders for the following:
- Specific strings
- Specific `.jar` files
- The names of files in a specific directory
- The names of files loaded by a specific class loader

Search type

Specifies the type of items in which to search for the string.

Table 38. Search type fields. Type searchable information in a field and click **Go**.

Search type	Instructions and resulting action
Class/Package	In the **Search terms** field, type a class name or package name. After you select this search type and click **Go**, the program searches class loaders for a class or package name. The program displays a list of classes and packages that have the string in their name.
JAR/Directory	In the **Search terms** field, type a `.jar` file name or directory name. After you select this search type and click **Go**, the program searches class loaders for a `.jar` file or directory name. The program displays a list of `.jar` files that have the string in their name and of all files in directories that have the string in their name.

Search terms

Specifies the string to be found in the items searched.

The search is case-sensitive. If the search string is `classname`, the string *ClassName* is not found.

The search matches the entire string. If the search type is **JAR/Directory** and the search string is `C:/WebSphere/AppServerd0603.185/java/jre/lib/ext/CmpCrmf.jar`, the entire path of the JAR file is matched. If the search type is **JAR/Directory** and the search string is `Cmp`, the string `Cmp` is not found.

The search supports limited regular expressions. It supports the wildcard characters asterisk (`*`), question mark (`?`), and percent sign (`%`). The wildcard characters `*` and `%` match zero or more characters; `?` matches exactly one character.

Table 39. Search strings with wildcard characters. Use the example search strings to see what items result from searches.

Search string	Resulting matches
Cmp	Items that have Cmp in their name
Cmp.jar	Items that have Cmp in their name and that end in .jar
%Cmp%	Items that have Cmp in their name
%Cmp%.jar	Items that have Cmp in their name and that end in .jar
*Cmp?rmf.jar	Items that have a name with any characters before Cmp, then any one character, and then rmf.jar

The search supports full regular expressions if the value for the search string starts and ends with a forward slash (/).

Table 40. Search strings with regular expressions. Use the example search strings to see what items result from searches.

Search string	Resulting matches
/.*Cmp.*/	Items that contain any character before and after Cmp in their name
/.*Cmp.*\.jar/	Items that have Cmp in their name and that end in .jar
/.*Cmp?rmf\.jar/	Items that have a name with any characters before Cmp, then any one character, and then rmf.jar
/.*\d\.jar/	Items with a name that ends in a number followed by .jar

Chapter 13. Choosing and using diagnosis tools and controls on z/OS

Below is a description of the types of tools and controls you can use for diagnosing and managing problems in the product environment.

Before you begin

The product uses a variety of different tools and server controls to help you collect specific types of data to determine where your servers are encountering problems. To efficiently use these tools you need to be aware of the different functions each can provide and what type of information will be available from each.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

About this task

When your applications or servers are experiencing problems that may be originating from different sources, use the tools below to collect data and information on processes in your environment. Each tool has functions specific to different parts of the product, and they can be used in concert to help you better diagnose your problems.

Procedure

Use the following z/OS tools to access and work with diagnostic information.

- **z/OS console**

  The console displays configuration errors that cause the termination of the product address spaces. Whatever goes to the console also goes to SYSLOG.
- **System log (SYSLOG)**

  SYSLOG is the repository for all messages that have appeared on the operator console. It also contains warning and informational messages that might be helpful after a failure has occurred.
- **Job log**

  The job log contains errors and warnings (non-termination) that are related to configuration. Anything that goes to the console and SYSLOG automatically goes to the job log.
- **System output (SYSOUT)**

  SYSOUT is a batch log that usually contains diagnostic data from the Java Virtual Machine (JVM) that runs in the servant. Any messages written to stderr will end up in SYSOUT. In addition, SYSOUT might contain error messages that usually appear in the log stream, but were redirected to SYSOUT, because the log stream was not available.
- **Error log**

  The error log contains messages issued through Java logging and JRas support, if any. In addition, the error log usually contains messages that are only intended for IBM use. These messages support actions, problems, or issues that are usually externalized through additional messages that are issued by other functions. When you work with IBM Support personnel, you might be asked to supply the error log so that service personnel can use these support messages to help diagnose the problem.

  Note: You must update the CFRM policy before using log streams that are CF-resident, such as the WebSphere error log and RRS logs. See Updating the CFRM policy for details.

- **SYSPRINT**

  SYSPRINT contains component trace (CTRACE) output for clients, and for servants when the product is configured to use SYSPRINT instead of CTRACE buffers and data sets.
- **Component trace (CTRACE) data set**

  CTRACE data sets contain diagnostic trace entries for various processes, depending on the trace options configured for the product.
- **Logrec**

  When an error occurs, the system records information about the error in the logrec data set or the logrec log stream. The information provides you with a history of all hardware failures, selected software errors, and selected system conditions.
- **Transaction XA Partner Log**

  This log is used for recovery of XA resources. When an application accesses XA resources, the product stores information about the resource to enable XA transaction recovery. For instructions on how to use the Profile Management Tool or the **zpmt** command to configure the Transaction XA Partner Log see the "Customization variables: Stand-alone application server cell" topic in the installing your application serving environment section. For instructions on how to change the location of the Transaction XA Partner Log, see the transaction service settings information.
- **SDSF**

  Use the SDSF DA panel to see how many application server address spaces are active, and observe at the CPU%, ECPU% and SIO rate. Use the "ENC" panel to see the enclaves running and what service classes they are running under.
- **RMF™**

  See Chapter 14, "Using RMF," on page 205 for instructions on starting and using RMF to monitor your transactions.
- **MODIFY command**

  See the getting help for the modify command example documentation for instructions on using the z/OS modify command to display information about the product servers or servants.

To find additional information about these tools, and about the process of diagnosing problems on z/OS, use the z/OS product library to access the following books:
- *z/OS MVS Diagnosis: Procedures, GA22-7587*, which helps you diagnose problems in the MVS operating system, its subsystems, its components, and in applications running under the system.
- *z/OS MVS Diagnosis: Tools and Service Aids, GA22-7589*, which provides detailed information about tools and service aids that can help you diagnose problems. This book contains a guide on how to select the appropriate tool or service aid for your purposes, and also provides an overview of all the tools and service aids available.

Troubleshooting using WebSphere variables

Troubleshooting problems can be performed by changing certain variables in your application environment.

Before you begin

WebSphere Application Server for z/OS provides configuration variables that control server behavior.
- Configuration variables may be set on a cell, node, or server level.
  - Variable values set on a cell level apply to all servers in all nodes in the cell, unless a different value for the same variable is set on a node or server level. Variable settings on a node or server level override values for the same variable set at the cell level.
  - Variables set on a node level apply to all servers in the node, unless a different value for the same variable is set on the server level. Variable settings on a server level override values for the same variable set at the node or cell level.
  - Variables set on a server level apply only to the specific server, not to any other servers in the same node or cell.

Note: When you are diagnosing particular problems, you are most likely to alter variable values on a server level, for a particular server. Specifying variable values on the server level affects both the controller and servant regions.

- You may use scripting interfaces, instead of the administrative console, to alter configuration variable values.
- These variables allow you to control:
  - Output destinations and characteristics for the error log, and for CTRACE buffers, data sets and the external writer.
  - Trace buffers, data sets, and the content of trace data.
  - Types of dumps to be requested.
  - Timeout values for system and application behavior.

About this task

Depending on the types of problems you encounter, you might need to change the values set for configuration variables that control WebSphere Application Server behavior. Generally speaking, the default values are designed for normal operation in a production environment. Other circumstances might require different values:

- When you first customize and verify WebSphere Application Server for z/OS installation, or
- When you test application workloads in a test environment, or
- when you encounter a problem, and need to collect more diagnostic data.

The following procedure explains how to use the administrative console to change configuration variable values, commonly known as console settings.

Procedure

1. Click **Environment -> Manage WebSphere Variables** in the console navigation tree.
2. On the **WebSphere Variables** page, select **Server** as the scope of the variable setting, and click **Apply**.
3. On the **WebSphere Variables** page, click **New**.
4. On the **Variable** page, specify a name and value for the variable. So other people can understand what the variable is used for, also specify a description for the variable. Then click **OK**.
5. Verify that the variable is shown in the list of variables.
6. Save your configuration.
7. To have the configuration take effect, stop the server and then start the server again.

Types of configuration variables

You can configure a variety of configuration variables to control the behavior of WebSphere Application Server for z/OS.

These configuration variables allow you to control:

- Output destinations and characteristics for the error log, and for CTRACE buffers, data sets and the external writer.
- Trace buffers, data sets, and the content of trace data.
- Types of dumps to be requested.
- Timeout values for system and application behavior.

Log output destinations and characteristics

Use these variables to control log output destinations and characteristics.

client_ras_logstreamname=*name:* Specifies the name of the log stream for an application client run-time to use for error information.

Default: If this variable is not specified, the client run-time uses SYSOUT.

Example:

```
client_ras_logstreamname=MY.CLIENT.ERROR.LOG
```

gotcha: Do not put the log stream name in quotes. Log stream names are not data set names.

ras_default_msg_dd=*DD_card_name:* Redirects write-to-operator (WTO) messages that use the default routing to hardcopy. These messages are redirected to the location identified through the DD card on the server's JCL start procedure. These WTO messages are primarily messages that WebSphere Application Server for z/OS issues during initialization.

Default: No default value is set; WTO messages that use default routing are sent to hardcopy.

Examples:

```
ras_default_msg_dd=MSGDD
ras_default_msg_dd=DFLTLOG
```

Example of the DFLTLOG DD card on the server's JCL start procedure:

```
//DFLTLOG    DD   SYSOUT=*
```

ras_hardcopy_msg_dd=*DD_card_name:* Redirects write-to-operator (WTO) messages that the product routes to hardcopy. These messages are redirected to the location identified through the DD card on the server's JCL start procedure. These WTO messages are primarily audit messages issued from Java code during initialization.

Default: No default value is set; WTO messages that use hardcopy routing are sent to hardcopy.

Example:

```
ras_hardcopy_msg_dd=MSGDD
```

ras_log_logstreamName: Specifies the log stream that the produce uses for error information during bootstrap processing. If the specified log stream is not found, or not accessible, a message is issued and errors are written to the server's job log.

Default: If this variable is not specified, the product uses SYSOUT.

Example:

```
ras_log_logstreamName=MY.CB.ERROR.LOG
```

gotcha: Do not put the log stream name in quotes. Log stream names are not data set names.

Trace control settings

The following trace options allow you to capture the information needed to detect problems.

To view or set these options, use the WebSphere Application Server administrative console:

1. Select **Environment > WebSphere variables**.
2. Specify the variable name in the name field and specify the setting in the value field. You can also describe the setting in the description field on this tab.

ras_trace_outputLocation=SYSPRINT | BUFFER | TRCFILE
Specifies where you want trace records to be sent:

- To SYSPRINT
- To a memory buffer (BUFFER), the contents of which are later written to a CTRACE data set
- To a trace data set (TRCFILE) specified on the `TRCFILE DD` statement in the start procedure for the server.

For servers, you can specify one or more values, separated by a space. For clients, you can only specify SYSPRINT.

Defaults:

- For clients, SYSPRINT
- For all other processes, BUFFER

Example: `ras_trace_outputLocation=SYSPRINT BUFFER`

`ras_time_local=0 | 1`
Specifies whether timestamps in trace records use Greenwich Mean Time (GMT) or local time. This variable setting controls timestamp format in the error log, and in traces sent to SYSPRINT or TRCFILE DD.

Default: 0 (GMT)

Example: `ras_time_local=1` sets timestamps to local time.

`Daemon_ras_trace_ctraceParms=SUFFIX | MEMBER_NAME`
Identifies the CTRACE PARMLIB member. The value can be either:

- A two-character suffix, which is added to the string `CTIBBO` to form the name of the PARMLIB member, or
- The fully specified name of the PARMLIB member. A fully specified name must conform to the naming requirements for a CTRACE PARMLIB member.

If the specified PARMLIB member is not found, tracing is defined to CTRACE, but there is no connection to a CTRACE external writer.

Note: The Daemon is the only server that recognizes this environment variable.

Default: None

Example: `Daemon_ras_trace_ctraceParms=01` identifies PARMLIB member CTIBBO01

`ras_trace_BufferCount=` *`n`*
Specifies the number of trace buffers to allocate. Valid values are 4 through 8.

Default: 4

Example: `ras_trace_BufferCount=6`

`ras_trace_BufferSize=` *`n`*
Specifies the size of a single trace buffer in bytes. You can use the letters K (for kilobytes) or M (for megabytes). Valid values are 128K through 4M.

Default: 1M

Example: `ras_trace_BufferSize=2M`

`ras_trace_log_version=` *`n`*
Specifies the version of trace log to display. Valid values are 1 and 2.

Default: 2

Example: `ras_trace_log_version=1`

Trace log stream record output

This article provides an example of the trace log stream output and explains the various attributes it contains.

If you do not want the message tag to be included in the trace log output, complete the following actions:

1. In the administrative console, click **Environment > WebSphere variables**.
2. Select the appropriate scope, and then click **New**.

3. Enter `ras_trace_log_version` in the **Name** field and `1` in the **Value** field.
4. Save and synchronize your changes, and then stop, and restart the server.

Sample output from the trace log: The numbers to the left of each sample were added to specify lines. The numbers will not be in the actual output.

```
1| Trace: 2009/07/14 17:26:19.577 02 t=6C8B58 c=UNK key=P8 tag=jperf
  (13007002)
2|  ThreadId: 0000002d
3|  FunctionName: PingServlet
4|  SourceId: PingServlet
5|  Category: AUDIT
6|  ExtendedMessage: BBOO0222I: Audit Message from PingServlet
```

The log stream record output fields from stream `BBO.BOSSXXXX` are:

Table 41. Parts table for a server log stream record output. Components and descriptions.

Component	Description
line 1: `Trace: 2009/07/14 17:26:19.577 02`	Date / timestamp / 2-digit record version number
line 1: `t=6C8B58`	Thread address
line 1: `c=UNK`	Cell name
line 1: `key=P8`	System protection key
line 1: `tag=jperf`	Message tag from classification file
line 1: `(13007002)`	Trace specific value for this trace point
line 2: `ThreadId: 0000002d`	Thread ID
line 3: `FunctionName:PingServlet`	Function Name
line 4: `SourceId:PingServlet`	Source ID
line 5: `Category:AUDIT`	Category
line 6: `ExtendedMessage: BBOO0222I: Audit Message from PingServlet`	Extended Message

Attention: Each field is delimited by a blank.

Dump control settings

Use these settings to manage the dump control configuration in WebSphere Application Server.

ras_dumpoptions_dumptype= *n*

Specifies the default dump used by the signal handler. Valid values and their meanings are:

- 0

  No dump is generated.
- 1

  A ctrace dump is taken.
- 2

  A cdump dump is taken.
- 3

  A csnap dump is taken.
- 4

  A CEE3DMP dump is taken.

  Note: CEE3DMP dumps are not available in WebSphere Application Server for z/OS with 64-bit support. If this option is chosen, it will be ignored in 64-bit environments.

CEE3DMP generates a dump of Language Environment® and the member language libraries. Sections of the dump are selectively included, depending on dump options specified, either by default or through the

```
ras_dumpoptions_ledumpoptons
```

variable. By default, this value passes

```
THREAD(ALL) BLOCKS
```

to CEE3DMP. You can override the default options for CEE3DMP through the

```
ras_dumpoptions_ledumpoptons
```

variable. For more information about CEE3DMP and its options, see z/OS Language Environment Programming Reference, SA22-7562..

Default: 3

Example:

```
ras_dumpoptions_dumptype=2
```

ras_dumpoptions_ledumpoptons= *options*

Specifies dump options to be used with a CEE3DMP. If you want more than one option, separate each option with a blank. Specifies dump options to be used with a CEE3DMP. If you want more than one option, separate each option with a blank.

This WebSphere variable is used only when you specify

```
ras_dumpoptions_dumptype=4
```

. For an explanation of CEE3DMP and valid dump options, see z/OS Language Environment Programming Reference, SA22-7562.

Rule: The maximum length of the option string on this environment variable is 255. If the option string is longer than 255, you receive message BBOM0011W and the CEE3DMP dump options are set to

```
THREAD(ALL) BLOCKS
```

.

Default:

```
THREAD(ALL) BLOCKS
```

Example:

```
ras_dumpoptions_ledumpoptons=NOTRACEBACK NOFILES
```

Timeout properties summary

You can use timeout properties to control the amount of time you allow for various requests to complete. Some of these properties map to internal variable names. The internal variable names are provided here to aid you with debugging.

Timer properties as they relate to configuring your message-driven beans to work with listener ports or activation specifications

For WebSphere Application Server Version 7 and later, listener ports are deprecated. Therefore plan to migrate your WebSphere MQ message-driven bean deployment configurations from using listener ports to using activation specifications. However, do not begin this migration until you are sure that the application does not have to work on application servers earlier than WebSphere Application Server Version 7. In some cases, you continue to use the WebSphere MQ message-driven bean deployment and listener ports and in other case you use the WebSphere MQ message-driven bean deployment and activation specifications.

The following properties DO NOT apply to activation specification driven message-driven bean deployment. That is, the properties require you to configure them to use the WebSphere MQ message-driven bean deployment and listener ports:

- control_region_mdb_request_timeout
- control_region_mdb_queue_timeout_percent
- server_region_mdb_stalled_thread_dump_action

The follow properties DO apply to activation specification driven message-bean deployment. That is, these properties require you to configure them to use the WebSphere MQ message-driven bean deployment and activation specifications.

- control_region_wlm_dispatch_timeout
- control_region_iiop_queue_timeout_percent
- server_region_iiop_stalled_thread_dump_action

As you follow the instructions to configure these properties, remember what properties apply to listener ports versus activation specifications.

Object Request Broker (ORB) service advanced settings

`ORB listener keep alive`

In a non-secure socket layer (SSL) environment, this property defines the value, in seconds, that is provided to TCP/IP on the SOCK_TCP_KEEPALIVE option for the IIOP listener. The function of this option is to verify if idle sessions are still valid by polling the client TCP/IP stack. If the client does not respond, then the session is closed. If the connection to the client is lost without the server receiving notification, then the session remains active on the server side. Use this option to clean up these unnecessary sessions.

- If this property is not set, then the TCP/IP option is not set.
- Setting the SOCK_TCP_KEEPALIVE option generates network traffic on idle sessions, which can be undesirable.

Default: 0

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers** > *server_name* **> Container services > ORB service > z/OS additional settings.**

`ORB SSL listener keep alive`

In an SSL environment, this property defines the value, in seconds, that is provided to TCP/IP on the SOCK_TCP_KEEPALIVE option for the IIOP listener. The function of this option is to verify if idle sessions are still valid by polling the client TCP/IP stack. If the client does not respond, then the session is closed. If the connection to the client is lost without the server receiving notification, then the session remains active on the server side. Use this option to clean up these unnecessary sessions.

- If this property is not set, then the TCP/IP option is not set.
- Setting the SOCK_TCP_KEEPALIVE option generates network traffic on idle sessions, which can be undesirable.

Default: 0

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers** > *server_name* **> Container services > ORB service > z/OS additional settings.**

`WLM timeout`

Specifies the maximum amount of time, in seconds, that workload management (WLM) waits for IIOP requests to complete. This time limit includes:

- The time during which the IIOP request waits on the WLM queue until being dispatched to a servant

- The time during which an application component, running in the servant, processes the request and generates a response

The server generates a failure response if this processing does not complete within the specified time.

Attention: This setting does not apply for HTTP requests or scalable messaging support; for that type of work, the value specified for the ConnectionResponseTimeout server custom property controls the time allowed for dispatching work to a servant.

Default: 300 seconds

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers** > *server_name* **> Container services > ORB service > z/OS additional settings.**

Internal variable name (for debugging purposes): Locate the internal variable name, control_region_wlm_dispatch_timeout, in the was.env file or the JES job log.

Example: `WLM timeout=600`

Use the control_region_iiop_queue_timeout_percent server custom property to designate a percentage of the WLM timeout as the amount of time a request can remain on the WLM queue.

`Request timeout`

Specifies, in seconds, the maximum time that the client waits for the response to a client request. The value specified for this field is a server wide setting that affects all outbound RMI/IIOP enterprise bean invocations that are made on this server.

Because the sysplex TCP/IP that runs through the coupling facility does not always tell the client when the other end of the socket has closed, clients can wait indefinitely for a response unless you set this property. Setting the **Request timeout** property ensures that the client gets a response within the specified time, even if the response is a COMM_FAILURE exception.

Default: 0 (unlimited). No timeout value is set.

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers** > *server_name* > **> Container services > ORB service > z/OS additional settings.**

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.RequestTimeout.

Example: Specifying `Request timeout=2`, sets the time limit to 2 seconds.

Transaction service timeout properties

`Total Transaction Lifetime Timeout`

Specifies the maximum amount of time, in seconds, that the J2EE server waits for an application transaction that originated in this server to complete if the application transaction does not set its own timeout value through the UserTransaction.setTransactionTimeout() method.

If the application transaction is not committed or rolled back within the specified time, the application transaction is marked for rollback and is allowed to continue running for a grace period of approximately 4 minutes. If the application transaction is committed or rolled back during the grace period, then the outcome of the transaction is always rolled back. If the application transaction does not complete after the grace period, then the controller abnormally ends the servant in which the application component is running with ABEND EC3 RSN=04130002 or 04130005.

gotcha: Only the total transaction lifetime timeout and the maximum transaction timeout have grace periods.

Setting this value to 0 indicates that the timeout does not apply, and the value of the maximum transaction timeout is used instead.

Default: 120 seconds

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers** > *server_name* > **Container services > Transaction service**.

Internal variable name (for debugging purposes): Locate transaction_defaultTimeout in the was.env file or the JES job log file.

Maximum transaction timeout

Specifies the maximum amount of time, in seconds, that the J2EE server waits for an application transaction that is propagated into this server to complete. This value also applies to transactions that are started in this server, if their associated applications do not set a transaction timeout and the total transaction lifetime timeout is set to `0`.

This value constrains the upper bound of all other timers. If an application uses the UserTransaction.setTransactionTimeout() method to specify a longer length of time, then the J2EE server changes the application setting to the value specified for the Maximum transaction timeout property.

Setting this value to 0 indicates that the timeout does not apply, and any transactions that are affected by this timeout never time out.

Default: 300 seconds

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers** > *server_name* > **Container services > Transaction service**.

Internal variable name (for debugging purposes): Locate the internal variable name, transaction_maximumTimeout, in the was.env file or the JES job log.

transaction_recoveryTimeout

Specifies the time, in minutes, that this controller uses to attempt to resolve in-doubt transactions before issuing a write-to-operator-with-reply (WTOR) message to the console that asks whether the controller should perform the following actions:

- Stop trying to resolve in-doubt transactions.
- Write transaction-related information to the job log or hardcopy log and terminate.

If the operator replies that recovery is to continue, then the controller attempts recovery for the specified amount of time before reissuing the WTOR message. After all the transactions are resolved, the controller region terminates. This property applies only to controllers in peer restart and recovery mode.

Default: 15 minutes

How to specify: To specify this property, in the administrative console, click **Environment > WebSphere variables**, select the appropriate node or cell from the list of available nodes and cells, and then click **New**. Add the transaction_recoveryTimeout property in the **Name** field, and specify a different value in the **Value** field.

Internal variable name (for debugging purposes): Locate transaction_recoveryTimeout in the was.env file or the JES job log.

Example: `transaction_recoveryTimeout=7`

Server custom properties

control_region_mdb_request_timeout

Specifies the time, in seconds, that the server waits for a message driven bean (MDB) request to receive a response. If the response is not received within the specified amount of time, then the servant might abnormally terminate with an EC3 ABEND, RSN=04130008. You can set this value to 0 if you need to disable this function.

Default: 120

How to specify: To specify this property, in the administrative console, click **Environment > WebSphere variables**, select the appropriate node or cell from the list of available nodes and cells, and then click **New**. Add the control_region_mdb_request_timeout property in the **Name** field, and specify a different value in the **Value** field.

Internal variable name (for debugging purposes): Locate control_region_mdb_request_timeout in the was.env file or the JES job log. See the application server z/OS custom properties documentation for additional information.

Example: `control_region_mdb_request_timeout=180`

Use the control_region_mdb_queue_timeout_percent server custom property name to designate a percentage of the value specified for the control_region_mdb_request_timeout property as the amount of time that a MDB request can remain on the WLM queue. The control_region_mdb_request_timeout custom property specifies the combined amount of time that the request spends on the WLM queue and in dispatch. The control_region_mdb_queue_timeout_percent property only applies to the amount of time that the request spends on the WLM queue.

`control_region_timeout_save_last_servant`

When set to `1`, this property indicates that, when the wlm_minimumSRCount custom property is set to a value that is greater than 1, then the last available servant is not abnormally terminated because of a timeout situation. The servant can be abnormally terminated after a new servant region starts to accept work requests. This setting enables work requests to continue without interruption. However, setting this property to `1` might cause a loss of system resources if the dispatched servant thread that timed out continues to loop or becomes inactive, preventing the servant threads assigned to this servant from being released.

This property can be set to `0` or `1`.

The setting for this property is ignored if the wlm_dynapplenv_single_server property is set to 1.

Default: 0

How to specify: To specify this property, in the administrative console, click **Environment > WebSphere variables**, select the appropriate node or cell from the list of available nodes and cells, and then click **New**. Add the control_region_timeout_save_last_servant property in the **Name** field, and specify `1` in the **Value** field.

Internal variable name (for debugging purposes): Locate control_region_timeout_save_last_servant in the was.env file or the JES job log.

`protocol_http_timeout_output_recovery`

Controls the recovery action taken on timeouts for requests received over the HTTP transport. Specifying `SERVANT` allows for the termination of servants when timeouts occur. If an HTTP request is under dispatch in a servant when its timeout value is reached, then the servant terminates with an ABEND EC3 RSN=04130007. The HTTP request and socket are then cleaned up. A setting of `SESSION` only cleans up the HTTP request and socket. No attempt is made to disrupt the processing of a dispatched HTTP request within a servant. Using the session setting might result in a loss of resources if the dispatched HTTP request loops or becomes inactive.

Default: SERVANT

How to specify: To specify this property, in the administrative console, click **Environment > WebSphere variables**, select the appropriate node or cell from the list of available nodes and cells, and then click **New**. Add the protocol_http_timeout_output_recovery property in the **Name** field, and specify a different value in the **Value** field.

Internal variable name (for debugging purposes): Locate protocol_http_timeout_output_recovery in the was.env file or the JES job log.

Example: `protocol_http_timeout_output_recovery=SERVANT`

protocol_https_timeout_output_recovery

Controls the recovery action taken on timeouts for requests received over the HTTPS transport. Specifying `SERVANT` allows for the termination of servants when timeouts occur. If an HTTP request is under dispatch in a servant when its timeout value is reached, then the servant terminates with an ABEND EC3 RSN=04130007. The HTTPS request and socket are then cleaned up. A setting of `SESSION` only cleans up the HTTPS request and socket. No attempt is made to disrupt the processing of a dispatched HTTPS request within a servant. Using the session setting might result in a loss of resources if the dispatched HTTPS request loops or becomes inactive.

Default: SERVANT

How to specify: To specify this property, in the administrative console, click **Environment > WebSphere variables**, select the appropriate node or cell from the list of available nodes and cells, and then click **New**. Add the protocol_https_timeout_output_recovery property in the **Name** field, and specify a different value in the **Value** field.

Internal variable name (for debugging purposes): Locate protocol_https_timeout_output_recovery in the was.env file or the JES job log.

Example: `protocol_https_timeout_output_recovery=SESSION`

protocol_sip_timeout_output

Specifies the time, in seconds, that the server waits for a message driven bean (MDB) request, that was sent over a SIP transport channel, to receive a response. If the response is not received within the specified amount of time, then the servant might abnormally terminate with ABEND EC3 RSN=04130008. You can set this value to 0 if you need to disable this function.

Default: 120

How to specify: To specify this property, in the administrative console, click **Environment > WebSphere variables**, select the appropriate node or cell from the list of available nodes and cells, and then click **New**. Add the protocol_sip_timeout_output custom property in the **Name** field, and specify a different value in the **Value** field.

Internal variable name (for debugging purposes): Locate protocol_sip_timeout_output in the was.env file or the JES job log.

Example: `protocol_sip_timeout_output=180`

Use the control_region_sip_queue_timeout_percent server custom property name to designate a percentage of the value specified for the protocol_sip_timeout_output property as the amount of time a request can remain on the WLM queue.

protocol_sips_timeout_output

Specifies the time, in seconds, that the server waits for a message driven bean (MDB) request to receive a response. If the response is not received within the specified amount of time, then the servant might abnormally terminate with ABEND EC3 RSN=04130008. Set this value to 0 to disable the function.

Default: 120

How to specify: To specify this property, in the administrative console, click **Environment > WebSphere variables**, select the appropriate node or cell from the list of available nodes and cells, and then click **New**. Add the protocol_sips_timeout_output custom property in the **Name** field, and specify a different value in the **Value** field.

Internal variable name (for debugging purposes): Locate protocol_sips_timeout_output in the was.env file or the JES job log. for additional information.

Example: `protocol_sips_timeout_output=180`

Use the control_region_sips_queue_timeout_percent server custom property name to designate a percentage of the value specified for the protocol_sips_timeout_output property as the amount of time a request can remain on the WLM queue.

protocol_sip_timeout_output_recovery
Controls the recovery action taken on timeouts for requests received over SIP. Specifying SERVANT allows for the termination of servants when timeouts occur. If a SIP request is under dispatch in a servant when its timeout value is reached, then the servant terminates with an ABEND EC3 RSN=04130007. The SIP request and socket are then cleaned up. A setting of SESSION only cleans up the SIP request and socket. No attempt is made to disrupt the processing of a dispatched SIP request within a servant. Using the session setting might result in a loss of resources if the dispatched SIP request loops or becomes inactive.

Default: SERVANT

How to specify: To specify this property, in the administrative console, click **Environment > WebSphere variables**, select the appropriate node or cell from the list of available nodes and cells, and then click **New**. Add the protocol_sip_timeout_output_recovery property in the **Name** field, and specify a different value in the **Value** field.

Internal variable name (for debugging purposes): Locate protocol_sip_timeout_output_recovery in the was.env file or the JES job log.

Example: `protocol_sip_timeout_output_recovery=SERVANT`

protocol_sips_timeout_output_recovery
Controls the recovery action taken on timeouts for requests received over SIPS. Specifying SERVANT allows for the termination of servants when timeouts occur. If an SIPS request is under dispatch in a servant when its timeout value is reached, then the servant terminates with an ABEND EC3 RSN=04130007. The SIPS request and socket are then cleaned up. A setting of SESSION only cleans up the SIPS request and socket. No attempt is made to disrupt the processing of a dispatched SIPS request within a servant. Using the session setting might result in a loss of resources if the dispatched SIPS request loops or becomes inactive.

Default: SERVANT

How to specify: To specify this property, in the administrative console, click **Environment > WebSphere variables**, select the appropriate node or cell from the list of available nodes and cells, and then click **New**. Add the protocol_sips_timeout_output_recovery property in the **Name** field, and specify a different value in the **Value** field.

Internal variable name (for debugging purposes): Locate protocol_sips_timeout_output_recovery in the was.env file or the JES job log.

Example: `protocol_sips_timeout_output_recovery=SERVANT`

server_region_request_cputimeused_limit
Specifies, in milliseconds, the amount of CPU time that an application request can consume.

This property helps prevent a single application request from monopolizing the available CPU time because it allows you to limit the amount of CPU time that a single request can use. A CPU monitor is invoked when a request is dispatched. If the request exceeds the specified amount of CPU time, the controller considers the request unresponsive. The controller then issues message BBOO0327, to let the requesting application know that the request was unresponsive.

The monitor, that monitors the amount of CPU time that a request is using, typically sends a signal to the dispatched thread when the amount of CPU time used exceeds the specified amount. However, there are situations when this signal cannot be delivered, and the request remains pending. For example, if the thread goes native and invokes a PC routine, the signal remains pending until the PC routine returns.

After the signal is delivered on the dispatch thread, the WLM enclave, that is associated with the dispatched request, is quiesced. This situation lowers the dispatch priority of this request, and this request should now only get CPU resources when the system is experiencing a light work load.

server_region_stalled_thread_threshold_percent
Specifies the percentage of threads that can become unresponsive before the controller terminates the

servant. When the default value of 0 is specified, the controller terminates the servant as soon as the controller determines that at least one thread has become unresponsive.

Default: 0

How to specify: To specify this property, in the administrative console, click **Environment > WebSphere variables**, select the appropriate node or cell from the list of available nodes and cells, and then click **New**. Add the server_region_stalled_thread_threshold_percent property in the **Name** field, and specify a different value in the **Value** field.

Internal variable name (for debugging purposes): Locate server_region_stalled_thread_threshold_percent in the was.env file or the JES job log.

Example: `server_region_stalled_thread_threshold_percent=5`

Java virtual machine (JVM) Custom properties

The following 2 JVM properties can be set by clicking **Application Servers -> <SERVER> -> Process Definition -> Servant -> Java Virtual Machine -> Custom Properties**.

`com.ibm.ws390.interrupt.disableBBOJ0122I`
If this property is set to 1, then message `BBOJ0122I` is suppressed.

Data Type: Boolean

Default: 0

Used by Daemon: No. Only applicable to servant regions.

`com.ibm.ws390.interrupt.applyDumpActionPreInterrupt`
Specifies the need to gather documentation prior to attempting to progress a dispatched request. If this property is set to 1, documentation specified by stalled_thread_dump_action is gathered prior to any activities to encourage the dispatched request to complete (for example, prior to driving interrupt() on any InterruptObject). The stalled_thread_dump_action defines which documentation to gather when the request is considered hung, and any attempts to complete it have failed.

Data Type: Boolean

Default: 0

Used by Daemon: No. Only applicable to servant regions.

Secure sockets layer configuration repertoires

depfeat: System SSL for z/OS has been deprecated in WebSphere Application Server Version 8.5. Start to convert any security scripts, that are based on System SSL, to use JSSE security.

`V3 Timeout`
Specifies the length of time, in seconds, that a browser can reuse a System SSL Version 3 session ID without renegotiating encryption keys with the server. The repertoires that you define for a server require the same V3 timeout value.

Default: 100

How to specify: To specify this property, in the administrative console, click **Security > SSL application servers > New SSL repertoire**

Internal variable name (for debugging purposes): The following SSL configuration repertoire timeout variables are set internally when you define your SSL repertoires:

- com_ibm_HTTP_claim_ssl_sys_v3_timeout
- com_ibm_DAEMON_claim_ssl_sys_v3_timeout

Locate these internal variables in the was.env file or the JES job log.

TCP transport channel timeout properties

Inactivity timeout property

Specifies the amount of time, in seconds, that the TCP transport channel waits for a read or write request to complete on a socket.

gotcha: The value specified for this property might be overridden by the wait times established for channels that are higher than this channel in the timer hierarchy. For example, the wait time established for an HTTP transport channel overrides the value specified for this property for every operation except the initial read on a new socket.

Default: 0 seconds

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers** > *server_name* **> Web container transport chains > TCP inbound channel**.

HTTP transport channel timeout properties

ConnectionResponseTimeout

Specifies a maximum amount of time, in seconds, that the server waits for an application component to respond to an HTTP request.

Set this property for each of the HTTP transport channel definitions on the server. You must set this property for both SSL transport channels and non-SSL transport channels. If the response is not received within the specified length of time, then the servant might fail with ABEND EC3 and RSN=04130007. Setting this timer prevents client applications from waiting for a response from an application component that might be in a deadlock, looping, or encountering some other processing problem that causes the application component to stop processing requests.

Default: 300 seconds

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name*, and then, under Web Container Settings, click **Custom properties**.

Internal variable name (for debugging purposes): If you are debugging a problem in SSL-enabled transport, then locate the internal variable name, protocol_https_timeout_output, in the was.env file or the JES job log. If you are debugging a problem in a non-SSL transport channel, then locate the internal variable name, protocol_http_timeout_output, in the was.env file or the JES job log.

Use the control_region_http_queue_timeout_percent and control_region_https_queue_timeout_percent application server custom properties to designate a percentage of the value specified for the ConnectionResponseTimeout property as the amount of time that a request can remain on the WLM queue.

Persistent timeout property

Specifies the amount of time, in seconds, that the HTTP transport channel allows a socket to remain idle between requests.

Default: 30 seconds

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name* **> Web container transport chains > HTTP inbound channel**.

Read timeout property

Specifies the amount of time, in seconds, that the HTTP transport channel waits for a read request to complete on a socket after the first read request occurs. The read that is completing might be an HTTP body, such as a POST, or part of the headers if the headers were not all read as part of the first read request on the socket.

Default: 60 seconds

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name* **> Web container transport chains > HTTP inbound channel**.

Write timeout property
Specifies the amount of time, in seconds, that the HTTP transport channel waits on a socket for each portion of response data to be transmitted. This timeout typically occurs in situations where responses lag behind new requests. This situation can occur when a client has a low data rate or the network interface card (NIC) for the server is saturated with I/O.

Default: 60 seconds

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name* **> Web container transport chains > HTTP inbound channel**.

HTTP transport timeout variables

depfeat: HTTP transport support is deprecated.

ConnectionIOTimeOut
Specifies a maximum amount of time, in seconds, that the J2EE server waits for the complete HTTP request to arrive. Set this property for each of the HTTP transport definitions on the server. You must set this property for both SSL transport and non-SSL transport. The J2EE server starts the timer after the connection has been established, and cancels the connection if a complete request does not arrive within the specified maximum time limit. Specifying a value of 0 disables the timeout function.

Default: 10 seconds

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers** > *server_name* > **Web container** > **Custom properties**.

gotcha: This panel is only available if an HTTP transport is defined for your application server environment. If an HTTP transport is not defined for your environment, then you can use the wsadmin scripting tool to define 1. However, it is recommended that you use an HTTP transport channel instead of an HTTP transport whenever possible.

ConnectionResponseTimeout
Specifies a maximum amount of time, in seconds, that the J2EE server waits for an application component to respond to an HTTP request. Set this property for each of the HTTP transport definitions on the server. You must set this property for both SSL transport and non-SSL transport. If the response is not received within the specified length of time, then the servant might fail with ABEND EC3 and RSN=04130007. Setting this timer prevents client applications from waiting for a response from an application component that might be in a deadlock, looping, or encountering some other processing problem that causes the application component to stop processing requests.

Default: 120 seconds

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers** > *server_name* > **Web container** > **Custom properties**.

Internal variable name (for debugging purposes): If you are debugging a problem in SSL-enabled transport, then locate the internal variable name, protocol_https_timeout_output, in the was.env file or the JES job log. If you are debugging a problem in non-SSL transport, then locate the internal variable name, protocol_http_timeout_output, in the was.env file or the JES job log.

Use the control_region_http_queue_timeout_percent and control_region_https_queue_timeout_percent server custom properties to designate a percentage of the ConnectionResponseTimeout property as the amount of time that a request can remain on the WLM queue.

ConnectionKeepAliveTimeout

Specifies the time, in seconds, that the J2EE server waits for a subsequent request from an HTTP client on a persistent connection. If another request is not received from the same client within this time limit, then the connection is closed.

Default: 30 seconds

How to specify: To specify this property, in the administrative console, click **Servers > Server Types > WebSphere application servers** > *server_name* > **Web container** > **Custom properties**.

Run-time environment: Best practices for maintaining the runtime environment

Use these guidelines to make sure that WebSphere Application Server for z/OS is customized and maintained correctly, to support your installation's application workload.

Checking these basic software and hardware requirements can help you avoid problems with the run-time environment.

- **Check that you have the necessary prerequisite software up and running.** Check that they have the proper authorizations and that the definitions are correct.
- **Check for messages that signal potential problems.** Look for warning and error messages in the following sources:
  - SYSLOG from other z/OS subsystems and products, such as TCP/IP (especially the DNS, if in use), RACF, and so on
  - WebSphere Application Server for z/OS error log
  - SYSPRINT of the WebSphere Application Server for z/OS
  - Component trace (CTRACE) output for the server
- **Check the ports used by WebSphere Application Server.** The ports that are used by WebSphere Application Server must not be reserved by any other z/OS component.
- **Ensure that z/OS has enough DASD space for SVC dumps.** You might have to adjust the amount of space, because it depends on the size of your applications, on the configured Java virtual machine (JVM) heap size, and on the number of servant regions that might be included in one dump, and so on. For an SVC dump of one controller and one servant, you can start with a minimum of 512, but might have to increase the MAXSPACE to 1024 or higher, given the factors listed above.
- **Check your general environment.** Does your system have enough memory? Insufficient memory problems can show up as AUX shortages, abends, or exceptions from the WebSphere Application Server for z/OS run-time. Sometimes the heap size for Language Environment (LE) and for the Java virtual machine (JVM) needs to be increased. If you are using RRS and DB2, make sure your system has enough space for archive data sets.
- **Make sure all prerequisite fixes have been installed**; a quick check for a fix can save hours of debugging.

  For the most current information on fixes and service updates, see:
  - The Preventive Service Planning (PSP) buckets for both WebSphere Application Server for z/OS and JAVA subsets of the WebSphere Application Server for z/OS Upgrade. To obtain a copy of the most current versions of these PSP buckets, you can either contact the IBM Support Center, use S/390® SoftwareXcel or link to IBMLink.
  - The Support web page of the WebSphere Application Server for z/OS website, which contains a table of the latest authorized program analysis reports (APARs).

  With the latest service information, check the following:
  - Ensure that all prerequisite PTFs (fixes) have been applied to the system.
  - Verify that all PTFs were actually present in the executables that were used at the time of error. Often, SMP can indicate that a fix is present and installed on the system when, in reality, the executables that were used at the time of error did not contain the fix.

System controls: Best practices for using system controls

Use this information as the best way to configure system controls.

- You have the option of using a z/OS system logger log stream as the product error log. The ras_log_logstreamName property identifies which log stream you want to use for the error log; it has no default setting. If you do not use a log stream, however, messages that usually appear in the error log are directed to server's job log.
- You have the option of directing trace output to SYSPRINT or buffers. The ras_trace_outputLocation property controls the location of trace output. The default values for this property are SYSPRINT for client applications, and buffers to all other processes. Although you can change the default for other processes from buffers to SYSPRINT, performance is better when you use buffers.
- You can use the Resource Measurement Facility (RMF) to view status information that might indicate potential problems. The product uses Workload Manager (WLM) services to report transaction begin-to-end response times and execution delay times, which might indicate that changes are required for timeout values or tuning controls.

Performance diagnosis information

The following report options are listed here for information. IBM Service may request that you run one or more of these reports while assisting you with diagnosis. You do not need to collect this data unless it is requested by IBM Service.

- If you suspect that you are having throughput problems in a particular address space, for example by looking at some other real-time performance data, IBM Service may need to see a dump of one or more address spaces. This is done using the following parameters:

```
JOBNAME=(<jobname list>)
SDATA=(LSQA,PSA,SQA,SUM,SWA,TRT,WLM,CSA,RGN)
```

- If you suspect that the problem could be resulting from GRS latch or ENQ contention, check the RMF Enqueue Activity Report and enter the console command:

```
D GRS,CONTENTION
```

  during a time period in which the performance problem is observed.
  SYS.BPX.A000.FSLIT.FILESYS.LSN represents HFS latches. Latch sets with a numeric suffix are file latches, specifically SYS.BPX.A000.FSLIT.FILESYS.LSN.01. If you detect file latch contention, the best way to determine the exact HFS file causing the problem is with an SVC dump, also collected during a time period in which contention occurred. You will need to dump one of the OMVS data spaces to get the file information.

```
DUMP COMM=(description of problem)
Reply to dump WTO, where serverproc is the name of your WebSphere Server
                   address space(s)
JOBNAME=(OMVS,Serverproc),DSPNAME=('OMVS'.SYSZBPX1,'OMVS'.SYSZBPX2),
SDATA=(CSA,GRSQ,LPA,NUC,PSA,RGN,SQA,TRT,SUM)
```

- Sometimes USS errors can cause performance problems. The USS Ctrace (SYSOMVS) MIN tracing option always records OMVS errors. You can take an SVC dump of the OMVS address space (as described in the previous bullet) and the data spaces and format the SYSOMVS CTRACE. Use IPCS options 7.2.1, suboption D, component SYSOMVS and the TALLY option (default is FULL). Look for trace events of errors in the TALLY report.
- To find delays in applications, collect application performance information
  - SMF 120 records.
  - Jinsight profile

Updating the CFRM policy

You must update the coupling facility resource management (CFRM) policy before using log streams that are CF-resident, such as the WebSphere Application Server error log and RRS logs. If you have the source for the current active CFRM policy, update the source and use the IXCMIAPU Administrative Data utility to generate the new policy.

About this task

If you do not have the source for the current active CFRM policy, rebuild the source from the active CFRM policy.

Procedure

1. Find the active policy by issuing the command: D XCF,POL You will get output similar to this (partial display):

```
D XCF,POL
  IXC364I  10.57.49  DISPLAY XCF 061
  . . .
  TYPE: CFRM
       POLNAME:       POLCF1N1
       STARTED:       03/14/2003 11:32:22
       LAST UPDATED:  03/14/2003 11:31:52
  . . .
```

2. List the active CFRM Policy's structure definitions by using the Administrative Data utility:

```
//STEP1   EXEC PGM=IXCMIAPU
//STDOUT DD   STDERR=*
//SYSABEND DD   STDERR=*
//SYSIN    DD   *
DATA TYPE(CFRM) REPORT(YES)
/*
```

3. Extract the definitions for the ACTIVE policy only. Using the STDOUT from the above utility job, edit the output with the following steps so it can be used to define a new policy in the next job:
   a. Extract the definitions for the ACTIVE policy only.
   b. Delete the heading lines.
   c. Add the new structure definition using the BBOWCFRM member of the target CNTL dataset as a model.
   d. Copy it into a FB-LRECL(80) dataset to be used as SYSIN for the following job.
4. Use the Administrative Data utility to update the CFRM policy. The policy name can be the same as the ACTIVE CFRM policy or a new name. If you use the active policy name, REPLACE(YES) must be specified on the DEFINE control statement.

```
//STEP20   EXEC PGM=IXCMIAPU
//STDOUT DD   STDERR=*
//SYSABEND DD   STDERR=*
//SYSIN    DD   *
 DATA TYPE(CFRM) REPORT(YES)

 DEFINE POLICY NAME(POLCF1N1) REPLACE(YES)

    CF NAME(CF1LPAR) DUMPSPACE(5000) PARTITION(0E) CPCID(00)
       TYPE(009672) MFG(IBM) PLANT(02) SEQUENCE(000000051205)

    CF NAME(CF2LPAR) DUMPSPACE(5000) PARTITION(0F) CPCID(00)
       TYPE(009672) MFG(IBM) PLANT(02) SEQUENCE(000000051205)

    STRUCTURE NAME(CTS130_DFHLOG) SIZE(24000) INITSIZE(12000)
       REBUILDPERCENT(1)  PREFLIST(CF1LPAR, CF2LPAR)
    . . .
              <== Insert your new structure definitions here
```

5. Activate the new policy by issuing the following MVS Command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=POLCF1N1
```

What to do next

For more information about coupling facility structures and the IXCMIAPU utility, see the z/OS manual MVS Setting Up a Sysplex (SA22-7625).

Error Dump and Cleanup interface

The Error Dump and Cleanup (BBORLEXT) interface exists to call WebSphere Application Server for z/OS in a recovery environment to allow it to request a dump and to clean up its resources.

The interface will:

- Save the function and DLL names of the failing z/OS component into the SDWA.
- Determine whether or not to issue an SDUMP, if relevant to the time-of-failure environment.
- Clean up z/OS internal structures and connections.

Program requirements: This interface **must** be called from within a WebSphere Application Server for z/OS location service daemon, controller (region), or servant (region). There are no restrictions against in which recovery environment, such as an ESTAE or FRR routine, the caller must reside.

General information

Interface	BALR to BBORLEXT
Address of routine	(ECVT+'234'x)+'20'x
Address mode	AMODE 31, RMODE any
State	Allow problem program state, task mode
Cross memory mode	PASN=HASN=SASN (non-cross memory)
Return codes	No return codes
Function	Clean-up various WebSphere for z/OS resources and possibly issue an SVC dump for the current address space

Input register information

1	Contains the address of the SDWA
14	Contains the return address
15	Contains the entry point address of BBORLEXT

Output register information

When control returns to the caller, the contents of the registers are as follows:

0-1	Used as a work register by the system
2-14	Unchanged
15	Used as a work register by the system

Note: Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service and restore them after the system returns control.

Note: A dump will not occur for X22 abends or for certain reason codes from 0D6, 052, 067, CC3, and DC3 abends. There may also be other error conditions that will not create a dump.

Example:

```
Example Here is an example of how to call this routine in assembler:
LA 1,SDWA        Load SDWA@ in Reg 1
L 15,(0,16)      Load CVT address
L 15,140(,15)    Load ECVT address
L 15,564(,15)    Load address of z/OS structure
L 15,32(,15)     Load address of z/OS routine
BALR 14,15       Invoke z/OS routine
```

Displaying information about current application server work

You can use either the administrative console or z/OS MVS console commands to accomplish multiple operator tasks that are related to application servers. The z/OS **display** or **modify** console commands can be used to obtain information about the work an application server is performing. You can also use these commands to perform tasks that are useful in diagnosing application server problems.

About this task

You can use the z/OS **display** or **modify** commands to perform the following actions:

- Set parameters that control application server operations.
- Display information about the work that an application server or servant is handling.
- Dynamically change values related to tracing activity for a server or servant.

With the **modify** command, you can display information about the following functions:

- Active controllers (servers)
- Servants
- Sessions
- Trace settings
- Java trace
- Java virtual machine (JVM) heap statistics
- Work elements
- Error logs

Procedure

1. Display the options for the **modify** command.

   You can use the help parameter to display the options that you can specify for the **modify** command. For example, entering the command, `f azsr01a,help`, generates information that is similar to the following information:

   ```
   BBOO0178I THE COMMAND MODIFY MAY BE FOLLOWED BY ONE OF THE FOLLOWING KEYWORDS:
   BBOO0179I CANCEL - CANCEL THIS CONTROL REGION
   BBOO0179I TRACEALL - SET OVERALL TRACE LEVEL
   BBOO0179I TRACEBASIC - SET BASIC TRACE COMPONENTS
   BBOO0179I TRACEDETAIL - SET DETAILED TRACE COMPONENTS
   BBOO0179I TRACESPECIFIC - SET SPECIFIC TRACE POINTS
   BBOO0179I TRACEINIT - RESET TO INITIAL TRACE SETTINGS
   BBOO0179I TRACENONE - TURN OFF ALL TRACING
   BBOO0179I TRACETOSYSPRINT - SEND TRACE OUTPUT TO SYSPRINT (YES/NO)
   BBOO0179I DISPLAY - DISPLAY STATUS
   BBOO0179I TRACE_EXCLUDE_SPECIFIC - EXCLUDE SPECIFIC TRACE POINTS
   BBOO0179I TRACEJAVA - SET JAVA TRACE OPTIONS
   BBOO0179I TRACETOTRCFILE - SEND TRACE OUTPUT TO TRCFILE (YES/NO)
   BBOO0179I MDBSTATS - MDB DETAILED STATISTICS
   BBOO0179I PAUSELISTENERS - PAUSE THE COMMUNICATION LISTENERS
   BBOO0179I RESUMELISTENERS - RESUME THE COMMUNICATION LISTENERS
   BBOO0179I TIMEOUTDUMPACTION - SET TIMEOUT DUMP ACTION
   BBOO0179I TIMEOUTDUMPACTIONSESSION - SET TIMEOUT DUMP ACTION SESSION
   BBOO0179I TRACETOTRCFILE - SEND TRACE OUTPUT TO TRCFILE DD CARD (YES/NO)
   ```

```
BBOO0179I MDBSTATS - MDB DETAILED STATISTICSBBOO0179I PAUSELISTENERS - PAUSE THE COMMUNICATION LISTENERS
BBOO0179I RESUMELISTENERS - RESUME THE COMMUNICATION LISTENERS
BBOO0179I STACKTRACE - LOG JAVA THREAD STACK TRACEBACKS
BBOO0179I TIMEOUTDUMPACTION - SET TIMEOUT DUMP ACTION
BBOO0179I TIMEOUTDUMPACTIONSESSION - SET TIMEOUT DUMP ACTION SESSION
BBOO0179I WLM_MIN_MAX - RESET WLM MIN/MAX SERVANT SETTINGS
```

2. Use the **display,help** parameter to display the parameters for the **DISPLAY** command.

   Entering the command, `f azsr01a,display,help`, generates information that is similar to the following information:

```
BBOO0178I THE COMMAND DISPLAY, MAY BE FOLLOWED BY ONE OF THE FOLLOWING KEYWORDS:
BBOO0179I SERVERS - DISPLAY ACTIVE CONTROL PROCESSES
BBOO0179I SERVANTS - DISPLAY SERVANT PROCESSES OWNED BY THIS CONTROL PROCESS
BBOO0179I SESSIONS - DISPLAY INFORMATION ABOUT COMMUNICATIONS SESSIONS
BBOO0179I TRACE - DISPLAY INFORMATION ABOUT TRACE SETTINGS
BBOO0179I JVMHEAP - DISPLAY JVM HEAP STATISTICS
BBOO0179I WORK - DISPLAY WORK ELEMENTS
BBOO0179I ERRLOG - DISPLAY THE LAST 10 ENTRIES IN THE ERROR LOG
BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,HELP
```

3. Use the z/OS **modifyserver,display,work** command to obtain information that might help diagnose problems, or to display application server information.

   You can enter the command, `f server_name,display,work,display_work_parameters`, where*server_name* is the name of the server about which you need to obtain information, and *display_work_parameters* is one of the following parameters:

   HELP
   Displays the **display,work** parameters.

   CLINFO
   Displays transaction classification information.

   EJB
   Displays IIOP driven Enterprise JavaBeans (EJB) requests, including total, current, dispatched and timed out.

   EJB,SRS
   Displays, by servant name, the IIOP driven EJB requests.

   SERVLET
   Displays HTTP driven servlet requests driven by HTTP, including total, current, dispatched and timed out.

   SERVLET,SRS
   Displays, by servant name, HTTP driven servlet requests.

   MDB
   Displays Java Message Service (JMS) driven message-driven bean (MDB) requests, including total, current, dispatched and timed out. Only messages received via MQ Message Listener Port (MLP) listeners in the controller are displayed.

   MDB,SRS
   Displays, by servant name, Java Message Service (JMS) driven message-driven bean (MDB) requests. Only messages received via MQ Message Listener Port (MLP) listeners in the controller are displayed.

   ALL
   Displays all of the preceding information for enterprise beans, servlets, and MDBs.

   ALL,SRS
   Displays, by servant name, all of the preceding information for enterprise beans, servants, and MDBs.

SUMMARY
Displays all of the current in-progress, and in-dispatch requests for enterprise beans, servlets, and MDBs.

SUMMARY,SRS
Displays, by servant name, all of the current in-progress, and in-dispatch requests for enterprise beans, servlets, and MDBs.

SIP
Displays the Session Initiation Protocol (SIP) requests driven, including total, current, dispatched and timed out.

SIP,SRS
Displays the SIP requests broken down by servant.

OLA
Displays inbound Optimized Local Adapter (OLA) requests driven by services BBOA1INV and BBOA1SRQ, including total, current, dispatched, and timed out.

OLA,SRS
Displays, by servant name, OLA requests driven by services BBOA1INV and BBOA1SRQ.

4. Display the content of the product error log using the display,errlog parameter of the **modify** command.

   The output from this command displays the last ten messages in the error log, even if you are not routing them to a log stream. For example, entering the command, `f x5sr01b,display,errlog`, generates information that is similar to the following information:

```
BBOO0266I (STC18876) BossLog: { 0001} 2003/11/25 20:08:55.120 01
SYSTEM=SYSB SERVER=X5SR01B  PID=0X010201B2 TID=0X12FB3F00 00000000
c=UNK ./bborjtr.cpp+812 ... BBOO0222I TRAS0017I: The startup trace
state is *=all=disabled.
BBOO0266I (STC18876) BossLog: { 0002} 2003/11/25 20:09:08.255 01
SYSTEM=SYSB SERVER=X5SR01B  PID=0X010201B2 TID=0X12FB3F00 00000000
c=UNK ./bborjtr.cpp+812 ... BBOO0222I SECJ0231I: The Security
component's FFDC Diagnostic Module com.ibm.ws.security.core.SecurityDM
registered successfully: true.
BBOO0266I (STC18876) BossLog: { 0003} 2003/11/25 20:09:09.562 01
SYSTEM=SYSB SERVER=X5SR01B  PID=0X010201B2 TID=0X12FB3F00 00000000
c=UNK ./bborjtr.cpp+812 ... BBOO0222I SECJ0212I: WCCM JAAS
configuration information successfully pushed to login provider class.
BBOO0266I (STC18876) BossLog: { 0004} 2003/11/25 20:09:09.573 01
SYSTEM=SYSB SERVER=X5SR01B  PID=0X010201B2 TID=0X12FB3F00 00000000
c=UNK ./bborjtr.cpp+812 ... BBOO0222I SECJ0240I: Security service
initialization completed successfully
BBOO0266I (STC18876) BossLog: { 0005} 2003/11/25 20:09:18.304 01
SYSTEM=SYSB SERVER=X5SR01B  PID=0X010201B2 TID=0X12FB3F00 00000000

c=UNK ./bborjtr.cpp+812 ... BBOO0223I PMI0023W: Unable to register
PMI module due to duplicate name: SoapConnectorThreadPool
BBOO0266I (STC18876) BossLog: { 0006} 2003/11/25 20:09:29.451 01
SYSTEM=SYSB SERVER=X5SR01B  PID=0X010201B2 TID=0X12FB3F00 00000000
c=UNK ./bborjtr.cpp+812 ... BBOO0222I SECJ0243I: Security service
started successfully
BBOO0266I (STC18876) BossLog: { 0007} 2003/11/25 20:09:29.464 01
SYSTEM=SYSB SERVER=X5SR01B  PID=0X010201B2 TID=0X12FB3F00 00000000
c=UNK ./bborjtr.cpp+812 ... BBOO0222I SECJ0210I: Security enabled false
BBOO0266I (STC18876) BossLog: { 0008} 2003/11/25 20:09:35.772 01
SYSTEM=SYSB SERVER=X5SR01B  PID=0X010201B2 TID=0X12FB3F00 00000000
c=UNK ./bborjtr.cpp+812 ... BBOO0223I PMI0023W: Unable to register PMI
module due to duplicate name: ProcessDiscovery
. . . .
BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,ERRLOG
```

5. Dynamically change values related to tracing activity for a server or servant using the z/OS **modify** command.

Table 42. z/OS modify command parameters and their equivalent WebSphere variables. The following table lists the **modify** *command parameters and the WebSphere variable that provides equivalent functionality.*

z/OS modify command parameter	Equivalent WebSphere variable	For more information, see
TRACEALL	ras_trace_defaultTracingLevel	Internal tracing tips for WebSphere for z/OS
TRACEBASIC	ras_trace_basic Do not change this variable unless directed to do so by IBM Support.	Setting trace controls for IBM service
TRACEDETAIL	ras_trace_detail Do not change this variable unless directed to do so by IBM Support.	Setting trace controls for IBM service
TRACESPECIFIC	ras_trace_specific Do not change this variable unless directed to do so by IBM Support.	Setting trace controls for IBM service
TRACE_EXCLUDE_SPECIFIC	ras_trace_exclude_specific Do not change this variable unless directed to do so by IBM Support.	Setting trace controls for IBM service
TRACEINIT	(no equivalent variable)	Example: Getting help for the modify command
TRACENONE	(no equivalent variable)	Example: Getting help for the modify command
TRACETOSYSPRINT	ras_trace_outputLocation=SYSPRINT	Internal tracing tips for WebSphere for z/OS
TRACETOTRCFILE	ras_trace_outputLocation=TRCFILE	Internal tracing tips for WebSphere for z/OS
TRACEJAVA	(no equivalent variable)	Example: Getting help for the modify command
TIMEOUTDUMPACTION	control_region_timeout_dump_action	Application server custom properties for z/OS
TIMEOUTDUMPACTIONSESSION	control_region_timeout_dump_action_session	Application server custom properties for z/OS

Example

The following command examples demonstrate how to use various **display,work** parameters and the resulting output. If you enter the command, `f azsr01a,display,work,all`, all of the information that is shown for each of the following individual commands is displayed.

```
f azsr01a,display,work,servlet

BBOO0255I TIME OF LAST WORK DISPLAY Wed Jan 3 19:17:54 2008
BBOO0256I TOTAL SERVLET REQUESTS         150670    (DELTA 1654)
BBOO0257I CURRENT SERVLET REQUESTS       1

BBOO0258I SERVLET REQUESTS IN DISPATCH   0
BBOO0267I TOTAL SERVLET TIMEOUTS         0    (DELTA 0)
BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,WORK,SERVLET

f azsr01a,display,work,servlet,srs

BBOO0255I TIME OF LAST WORK DISPLAY Wed Jan 3 19:18:01 2008
```

```
BBOO0259I STC18964:  TOTAL SERVLET REQUESTS  152344   (DELTA 1675)
BBOO0260I STC18964:  SERVLET REQUESTS IN DISPATCH    0
BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,WORK,SERVLET,ALL

 (EJB and MDB displays would look the same except for the request type.)

f azsr01a,display,work,summary

BBOO0255I TIME OF LAST WORK DISPLAY Wed Jan 3 19:18:38 2008
BBOO0261I TOTAL REQUESTS TO SERVER     173591    (DELTA 13944)
BBOO0262I TOTAL CURRENT REQUESTS       0
BBOO0263I TOTAL REQUESTS IN DISPATCH   0
BBOO0268I TOTAL TIMED OUT REQUESTS     0    (DELTA 0)
BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,WORK,SUMMARY

f azsr01a,display,work,summary,srs

BBOO0255I TIME OF LAST WORK DISPLAY Wed Dec  3 19:27:01 2003
BBOO0264I STC18964:  TOTAL REQUESTS       173591    (DELTA 0)
BBOO0265I STC18964:  TOTAL REQUESTS IN DISPATCH  0
BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,WORK,SUMMARY,SRS
```

Some points to consider:

- Adding `,help` at the end of the command displays your choices. For example, entering the command, `display,work,help`, displays the options that you can specify for the **display,work** command.
- Entering `display,work` produces the same output as entering `display,work,summary`.
- The **display,EJB** command only displays IIOP requests. Therefore, if an HTTP driven request runs a JavaServer page (JSP), and a servlet that calls an enterprise bean, only the servlet count increases, because only requests that are used to get to the controller are counted. The JSP does not use the controller.
- When displaying the work by servant, only the total and in-dispatch requests display. Without the SRS parameter, the current requests and timeouts also display.
- Work continues while the data is collected. The current data is saved and used to calculate the delta when the next command is issued. In a heavy load situation, the displayed values and deltas might not add up as you expect.
- Entering TOTAL displays all of the work items that were handled since the last **display,work** command was entered. This display does not include the current in-flight requests.

Chapter 14. Using RMF

RMF can usually be started with the simple 'S RMF' command from the MVS console.

About this task

The Monitor III data gatherer can be started after RMF with the 'F RMF,S III' modify command. To use the RMF Monitor 3 display, go to the “Sysplex Summary” display as follows:

Procedure

1. Type RMF on ISPF command line.
2. Type 3 to see Monitor III choices.
3. Type S to get Sysplex reports.
4. Type 1 to see the Sysplex Summary report. You can scroll back and forth in time with PF10 and PF11, or over-type the time field. Look at the transactions in the WebSphere Application Server service and reporting class and note the Average Response time, Transactions per second, and Performance Index. You may also explore further in RMF Monitor III to see the “System Information” report.

Example

Here is a typical RMF procedure:

```
//RMF      PROC
//IEFPROC  EXEC PGM=ERBMFMFC,REGION=0M,PARM='MEMBER(XS)'
//IEFPARM  DD   DDNAME=IEFRDER
//IEFRDER  DD   DSN=SYS1.PARMLIB,DISP=SHR
```

The following is a copy of the PARMLIB member ERBRMFXS: (parmeters beginning with a /* are not used in this example but may be useful to you.

```
    CPU                       /* COLLECT CPU STATISTICS        */
         CHAN                      /* COLLECT CHANNEL STATISTICS    */
         CYCLE(1000)               /* SAMPLE AT 1 TIME / SECOND     */
         DEVICE(NOCHRDR)           /* NO CHARACTER RDR DEV STATS    */
         DEVICE(COMM)              /* ADDED COMM FOR 37X5           */
         DEVICE(DASD)              /* COLLECT DIRECT ACCESS DEVICE  */
         /* STATISTICS                   */
         DEVICE(NOGRAPH)           /* NO GRAPHICS DEVICE STATISTICS */
         DEVICE(NOTAPE)            /* NO TAPE DEVICE STATISTICS     */
         DEVICE(NOUNITR)           /* NO UNIT RECORD DEVICE STATS   */
         ENQ(SUMMARY)              /* ENQ REPORTING                 */
         INTERVAL(15M)             /* REPORT AT  15 MIN INTERVALS   */
         IOQ(DASD)                 /* I/O Q'ING FOR DEV IN LOG CU   */
         IOQ(COMM)                 /* I/O Q'ING FOR DEV IN LOG CU   */
         NOVSTOR                   /* NO RMF 3.2 AND LATER REL */
         OPTIONS                   /* OPERATOR MAY CHG RMF OPTIONS  */
         PAGING                    /* COLLECT PAGING STATISTICS     */
         PAGESP                    /* COLLECT PAGE/SWAP DATASET STAT*/
         RECORD                    /* RECORD INTO SMF DATASET       */
         NOSTOP                    /* STOP AFTER  90 MINUTES        */
         SYNC(SMF)                 /* INTERVAL SYNCED WITH SMF      */

         STDERR(H)                 /* STDERR CLASS OF OUTPUT REPORT */
         WKLD(PERIOD,SYSTEM)       /* COLLECT WORKLOAD MANAGER
                                      STATISTICS AND REPORT AT THE
                                      PERIOD LEVEL + TOTAL LINE     */
        TRACE(CCVUTILP)
```

Chapter 15. Collecting job-related information with the System Management Facility (SMF)

SMF can be enabled to collect and record system and work-related information on the WebSphere for z/OS system. This information can be used to bill users, report system reliability, analyze your configuration, schedule work, identify system resource usage, and perform other performance-related tasks that your organization may require.

About this task

You can enable SMF recording for:

- Capacity planning, to determine:
  - How many transactions have run?
  - What is the average and maximum completion time for methods running on each server?
  - How many clients are attached to each server instance? Of these clients, how many are active?
- Application profiling:
  - To show an application broken down into its component parts.
  - To provide timing information on the application's component parts.
- Error reporting:
  - To detect and record soft failures (those that are generated through an exception or those that are performance-related).
  - To use this error information to trigger an event that will cause an action to occur once a threshold has been reached.

Procedure

- Read “Enabling SMF recording” on page 209 for information on enabling SMF type 120 records.
- Read “Viewing the output data set” on page 212 for steps on viewing the data you record.
- Read “Disabling SMF recording for WebSphere Application Server” on page 212 for steps on disabling SMF data collection.

Example

The SMF Browser available on the WebSphere for z/OS download site is able to display record type 120. To download the SMF Browser go to: https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=zosos390. For further information on the SMF Browser, download the browser package and read the associated documentation.

The following example shows sample output from the SMF Browser. The example features subtype 7 and subtype 8, in that order.

```
Record#: 14;
 Type: 120; Size: 820; Date: Fri Nov 23 04:54:17  EST 2001;
 SystemID: SY1; SubsystemID: WAS; Flag: 94;
 Subtype: 7 (WEB CONTAINER ACTIVITY);
# Triplets: 4;
Triplet #: 1; offset: 76; length: 32; count: 1;
Triplet #: 2; offset: 108; length: 140; count: 1;
Triplet #: 3; offset: 264; length: 556; count: 1;
Triplet #: 1; Type: ProductSection;
  Version: 1; Codeset: Unicode; Endian: 1; TimeStampFormat: 1 (S390STCK64);
  IndexOfThisRecord: 1; Total # records: 1; Total # triplets: 4;
Triplet #: 2; Type: WebContainerActivitySection;
  HostName : PLEX1;
  ServerName : BBOASR4;
  ServerInstanceName: BBOASR4A;
  WlmEnclaveToken * 00000020 00000242 -------- --------
                  * ^... * p1047
```

```
  ActivityID * b6c7a7b7 14e9bc85 000000b0 00000007
             * 0926306b -------- -------- --------
             *,............ *Cp1047
             ActivityStartTime * b6c7a7b7 14e9bc85 40404040 40404040 *
             ActivityStopTime * b6c7a7b7 53a8a645 40404040 40404040 *
Triplet #: 3; Type: HttpSessionManagerActivitySection;
  # http sessions created: 0; # http sessions invalidated: 0;
  # http sessions active: 0;
Average session life time: 0 [sec*10**-3];
 Triplet #: 4; Type: WebApplicationActivitySection;
  Name: PolicyIVP-localhost_1;
  # Servlets: 1;
  Triplet #: 4.1; offset: 272; length: 284; count: 1;
  Triplet #: 4.1; Type: ServletActivitySection;
   Name: SimpleFileServlet;
   ResponseTime: 48 [sec*10**-3];
   # errors: 0;
   Loaded by this request: 0;
   Loaded since (raw): ea54948e0d;
   Loaded since: Thu Nov 22 10:02:49 EST 2001;
Record#: 72;
  Type: 120; Size: 1744; Date: Fri Nov 23 05:01:02 EST 2001;
  SystemID: SY1; SubsystemID: WAS; Flag: 94;
  Subtype: 8 (WEB CONTAINER INTERVAL);
  # Triplets: 4;
  Triplet #: 1; offset: 76; length: 32; count: 1;
  Triplet #: 2; offset: 108; length: 112; count: 1;
  Triplet #: 3; offset: 264; length: 1480; count: 1;
  Triplet #: 1; Type: ProductSection;
    Version: 1; Codeset: Unicode; Endian: 1; TimeStampFormat: 1 (S390STCK64);
    IndexOfThisRecord: 1; Total # records: 1; Total # triplets: 4;
  Triplet #: 2; Type: WebContainerIntervalSection;
    HostName : PLEX1;
    ServerName : BBOASR4;
    ServerInstanceName: BBOASR4A;
    SampleStartTime * b6c7a6fd 655c0604 40404040 40404040 *
    SampleStopTime * b6c7a939 9a0e614c 40404040 40404040 *
Triplet #: 3; Type: HttpSessionManagerIntervalSection;
    http sessions #created: 1; #invalidated: 0;
    http sessions #active: 0; Min #active: 0; Max #active: 0;
    Average session life time: 0;
    Average session invalidate time: 0;
    http sessions #finalized: 0; #tracked: 0;
    http sessions #min live: 0; #max live: 0;
  Triplet #: 4; Type: WebApplicationIntervalSection;
    Name: PolicyIVP-localhost_1;
    # Servlets loaded: 0;
    # Servlets: 4;
    Triplet #: 4.1; offset: 312; length: 292; count: 1;
    Triplet #: 4.2; offset: 604; length: 292; count: 1;
    Triplet #: 4.3; offset: 896; length: 292; count: 1;
    Triplet #: 4.4; offset: 1188; length: 292; count: 1;
    Triplet #: 4.1; Type: ServletIntervalSection;
      Name: SimpleFileServlet;
      # requests: 6;
      AverageResponseTime: 764 [sec*10**-3];
      MinimumResponseTime: 18 [sec*10**-3];
      MaximumResponseTime: 4133 [sec*10**-3];
      # errors: 0;
      Loaded since (raw): ea54948e0d;
      Loaded since: Thu Nov 22 10:02:49 EST 2001;
    Triplet #: 4.2; Type: ServletIntervalSection;
      Name: Was40Ivp;
      # requests: 4;
      AverageResponseTime: 4664 [sec*10**-3];
      MinimumResponseTime: 1584 [sec*10**-3];
      MaximumResponseTime: 12572 [sec*10**-3];
      # errors: 0;
      Loaded since (raw): ea58a1509e;
      Loaded since: Fri Nov 23 04:55:14 EST 2001;
    Triplet #: 4.3; Type: ServletIntervalSection;
      Name: /cebit.jsp;
```

```
  # requests: 1;
  AverageResponseTime: 204 [sec*10**-3];
  MinimumResponseTime: 204 [sec*10**-3];
  MaximumResponseTime: 204 [sec*10**-3];
  # errors: 0;
  Loaded since (raw): ea58a24a69;
  Loaded since: Fri Nov 23 04:56:18 EST 2001;
Triplet #: 4.4; Type: ServletIntervalSection;
  Name: JSP 1.1 Processor;
  # requests: 1;
  AverageResponseTime: 482 [sec*10**-3];
  MinimumResponseTime: 482 [sec*10**-3];
  MaximumResponseTime: 482 [sec*10**-3];
  # errors: 0;
  Loaded since (raw): ea54948b66;
  Loaded since: Thu Nov 22 10:02:48 EST 2001;
```

Enabling SMF recording

Use this page to enable SMF recording for WebSphere Application Server and select SMF type 120 records for output to the SMF data sets.

About this task

For an overview of SMF recording, see Chapter 1 of z/OS MVS System Management Facilities (SA22-7630)

To enable properties for specific record types, use the administrative console.

.

Procedure

1. Edit the SMFPRMxx parmlib member and update the SYS or SUBSYS(STC,...) statement to include the type 120 record.

   ```
   SUBSYS(STC,EXITS(IEFU29,IEFACTRT),INTERVAL(SMF,SYNC), TYPE(0,30,70:79,88,89,120,245))
   ```

2. Use the SET command to indicate which SMF parmlib member the system should use. You must issue the SET command before you start WebSphere Application Server. If you issue the command after the application server has started, SMF type 120 records will not be collected.
3. Enable SMF type 120 records.
4. Format the output data set.

Results

You have successfully enabled SMF recording when the SMF data is recorded in the data set which is specified in SMFPRMxx.

Using the administrative console to enable properties for specific SMF record types

You can use the administrative console to view or set SMF record properties for specific types of SMF records.

Before you begin

Ensure that you have proper access to the administrative console.

About this task

To enable SMF, you must first use the administrative console to enable properties for specific record types.

Procedure

1. Go to the Environment entries page of the administrative console.

   Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name*** > **Java and Process Management** > **Process definition** > **Environment entries**.
2. To enable SMF type 120 records, click **New**, and specify one or more of the following properties:
   - name = server_SMF_server_activity_enabled = 1 (or server_SMF_server_activity_enabled = true)
   - name = server_SMF_server_interval_enabled = 1 (or true)
   - name = server_SMF_container_activity_enabled = 1 (or true)
   - name = server_SMF_container_interval_enabled = 1 (or true)
   - name = server_SMF_interval_length, value=n, where n is the interval, in seconds, that the system will use to write records for a server instance. Set this value to 0 to use the default SMF recording interval.
   - name = server_SMF_request_activity_enabled = 1 (or true)
   - name = server_SMF_request_activity_CPU_detail = 0. To enable the property, set the value to 1.
   - name = server_SMF_request_activity_timestamps = 0 To enable the property, set the value to 1.
   - name = server_SMF_request_activity_security = 0 To enable the property, set the value to 1.
   - name = server_SMF_request_activity_async = 0 To enable the property, set the value to 1.
   - name = server_SMF_outbound_enabled = 1 (or true)
3. Click **OK** or **Apply**.
4. Save the changes and make sure a file synchronization is performed before restarting the servers.
5. For the changes to take effect, restart the application server.

Results

You have successfully activated SMF recording for the product when SMF type 120 records are being recorded.

Editing the SMFPRMxx parmlib member

Use this page to edit the SMFPRMxx parmlib member and enable SMF recording for WebSphere Application Server for z/OS.

Before you begin

Make a working copy of the sample PARMLIB member SMFPRMYL.

About this task

Follow these steps to edit the SMFPRMxx parmlib member and enable SMF recording for WebSphere Application Server:

Procedure

1. Insert an **ACTIVE** statement to indicate SMF recording. See *z/OS MVS Initialization and Tuning Guide* for more information.
2. Insert a **SYS** statement to indicate the types of SMF records you want the system to create. For example, use SYS(TYPE(120:120)) to select WebSphere Application Server type 120 records only. Keep the number of selected record types small to minimize the performance impact.

3. You can specify the interval in which you want the Server and Container interval records created (if no interval was specified in administrative console for the server or container definition) using the **INTVAL (mm)** statement in the SMFPRMxx parmlib member . The default SMF recording interval is 30 minutes. See *z/OS MVS Initialization and Tuning Reference* for more information.

   The server and container interval records will use either:

   - The value specified in the server/container definition as specified in the administrative console
   - The interval specified in the SMF parmlib member (from the SMF product settings) if you specify a length of 0.

Writing records to DASD

You can configure WebSphere Application Server for z/OS to write records to DASD locations.

Before you begin

Make sure you have your modified PARMLIB member SMFPRMxx.

About this task

Follow this step to start writing records to DASD:

Procedure

Issue the following command: **t smf=xx** where xx is the suffix of the SMF parmlib member (SMFPRMxx). See *z/OS MVS System Management Facilities (SMF), SA22-7630* for more information.

Results

Writing records to DASD has been completed successfully when the data is recorded in the data set which is specified in SMFPRMxx.

Formatting the output data set

Use this page to format the SMF recording output data set into a readable format for printing to the screen or other output device.

Before you begin

Make sure SMF recording is running.

About this task

Perform the following steps to format the SMF recording output data set into a readable format for printing to the screen or other output device:

Procedure

1. Switch the SMF data sets by entering **i smf** from the MVS console to switch the SMF data sets.
2. Run the SMF Dump program (IFASMFDP) to create a sequential data set from the raw dump. A sample JCL is shown in *z/OS MVS System Management Facilities (SMF), SA22-7630*.

Results

You have successfully formatted the output data set when SMFDUMP ends with return code 0.

Viewing the output data set

Use the SMF Browser to view the output data set from the WebSphere Application Server for z/OS UNIX environment.

Before you begin

The data set should be viewed using a program that can display record type 120.

About this task

The SMF Browser is provided in the form of a JAR file named `bbomsmfv.jar`. The `bbomsmfv.jar` file is not supported by IBM. To use it from the WebSphere Application Server for z/OS UNIX environment:

Procedure

1. Verify that the *JAVA_HOME* environment variable refers to the current Java installation: JAVA_HOME=../usr/bin/java/J1.3. This must be at least Java 1.3 since this release is the first to implicitly contain the necessary record support needed by the SMF Browser.
2. Download the SMF Browser from the WebSphere Application Server for z/OS website at: https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=zosos390.
   The website also provides a separate `batchsmf.jar` file to format batch data.
3. Copy the `bbomsmfv.jar` file to your tools directory. Be sure that any edits made to the file in the future are made to both copies of the file, or run from the installation directory in the first place.
4. To interpret SMF data from a cataloged WebSphere Application Server for z/OS sequential file named "USER.SMFDATA" (which was previously created using the IFASMFDP utility as described previously), run: `java -cp bbomsmfv.jar com.ibm.ws390.sm.smfview.SMF "INFILE(USER.SMFDATA)"`. It is implicit in the Java command parameterization that your current working directory is the tools directory. If this is not the case, you receive a NoClassDefFoundError on com.ibm.ws390.sm.smfview.SMF message. Java does not generate a diagnostic when it does not find the `bbomsmfv.jar` file in the current directory.
   A sample report is provided in Collecting job-related information with the System Management Facility (SMF).
   You can create a summary report with the `PLUGIN` parameter. For example, for a batch summary report:

   ```
   java -cp bbomsmfv.jar:batchsmf.jar com.ibm.ws390.sm.smfview.SMF 'INFILE(USER.SMFDATA)' 'PLUGIN(PERFSUM,STDOUT)'
   ```

   The `PLUGIN` keyword requires a class (`DEFAULT` or `PERFSUM`) and parameter string (`STDOUT` or file name, such as `/tmp/SmfOutput.txt`) separated by a comma.

Results

The SMF Browser is successfully installed and invoked when you do not receive any Java error messages after the invocation and the Browser output is shown on the screen.

Disabling SMF recording for WebSphere Application Server

This information describes how to disable SMF recording for WebSphere Application Server for z/OS.

Before you begin

Ensure that you have proper access to the administrative console.

About this task

SMF recording can be enabled for WebSphere Application Server, and for z/OS. The following steps describe how to disable SMF recording for WebSphere Application Server:

Procedure

1. In the administrative console, click **Servers > Application Servers** > *server_name*.
2. On the configuration page, click **Server Infrastructure > Java and Process Management > Process Definition > Control > Environment Entries**.
3. To disable SMF type 120 records, set the following properties to false:
   - name = server_SMF_server_activity_enabled = 0 (or server_SMF_server_activity_enabled = false)
   - name = server_SMF_server_interval_enabled = 0 (or false)
   - name = server_SMF_container_activity_enabled = 0 (or false)
   - name = server_SMF_container_interval_enabled = 0 (or false)
   - name = server_SMF_request_activity_enabled = 0 (or false).
   - name = server_SMF_request_activity_CPU_detail = 0. To enable the property, set the value to 1.
   - name = server_SMF_request_activity_timestamps = 0 To enable the property, set the value to 1.
   - name = server_SMF_request_activity_security = 0 To enable the property, set the value to 1.
   - name = server_SMF_request_activity_enabled=0 (or server_SMF_request_activity_enabled=false).
   - name = server_SMF_request_activity_async = 0 (or false)
   - name = server_SMF_outbound_enabled = 0 (or false)

   Alternatively, you could delete the SMF related properties. However, it will be easier for you to enable SMF recording later if you keep the properties in place and just change their values to false.
4. Click **OK** or **Apply**.
5. Save the changes and make sure a file sync is performed before restarting the servers.
6. For the changes to take effect, restart the application server.

Results

You have successfully disabled SMF recording for WebSphere Application Server when SMF records of records type 120 are no longer being recorded.

Disabling SMF recording for the entire MVS system

Use this page to disable SMF recording for your MVS System (z/OS).

Before you begin

Make sure that you have your own working copy of SMFPRMxx and SMF is running.

About this task

SMF recording can be enabled for WebSphere Application Server and for z/OS. The following steps describe how to disable SMF recording for your MVS System (z/OS):

Procedure

Edit the SMFPRMxx parmlib member and set SMFPRMxx to "NOACTIVE" which will disable the writing of SMF records to DASD. Use the SET command to activate that SMF parmlib member on the MVS system.

Results

SMF recording has successfully been disabled for the whole MVS system when SMF records for z/OS and WebSphere Application Server are no longer being written to DASD.

Using SMF type 80 - preparing for audit support

SMF type 80 requires some preparation in order to be fully utilized in a WebSphere environment.

Before you begin

As WebSphere Application Server becomes more capable of authentication and setting or changing the identity on a thread, so arises the need for the ability to audit these changes. Along with this also comes the need to audit the accompanying authorization requests made through EJBRoles checking, intending to produce audit records that include the original authenticated identity. This auditing in WebSphere Application Server is managed not through WebSphere Application Server itself, but through its External Security Manager (RACF or equivalent), where the SMF records are cut.

About this task

In order to take advantage of auditing in WebSphere Application Server, you need to set up SMF and RACF and have both running.

Procedure

1. Set up SMF for audit support. For information on setting up and starting SMF, see *z/OS MVS System Management Facilities (SMF), SA22-7630*
2. Enable auditing for the EJB Roles by setting the RACF AUDIT attribute. This will set up RACF for auditing in WebSphere Application Server. You can turn on auditing for the ADMIN and PAYROLL classes with the following command:

   ```
   RALTER EJBROLE (ADMIN,PAYROLL) AUDIT(ALL)
   ```

   - Alternately, you could modify the RACFROLE job to put the AUDIT information there.
   - For more information and additional parameters for the AUDIT attribute, see the *z/OS Security Server RACF Auditor's Guide*.

Audit support

This topic gives an overview of how to use audit support.

Auditing is performed using SMF records issued by RACF or an equivalent External Security Manager. This means that SMF audit records are cut as part of the WebSphere Application Server use of SAF interfaces such as IRRSIA00 (to manage ACEEs) and the RACROUTE macro.

Table 43. Security authentication mechanisms and the corresponding data that is written to each part of the ACEE X500NAME field. The following table lists the various security authentication mechanisms and the corresponding data that is written to each part of the ACEE X500NAME field (this data is also in the RACO and SMF records).

Authentication mechanism	Service name	Authenticated identity
Custom Registry	WebSphere Custom Registry	Custom registry principal name
Kerberos	Kerberos for WebSphere Application Server	Kerberos principal, in the "DCE" format used for extracting the corresponding MVS userid using IRRSIM00 (/.../realm/principal)
RunAs Rolename	WebSphere Role Name	Role name
RunAs Server	WebSphere Server Credential	MVS userid
Trust Interceptor	WebSphere Authorized Login	MVS userid
RunAs Userid/Password	WebSphere Userid/Password	MVS Userid

In addition to tracking by MVS userid, events need to be traced to an originating userid. This is especially true for originating userids that are not MVS-based, such as EJB Roles, Kerberos principals, and Custom Registry principals.

SMF settings

Configure SMF records to collect job information to tune application server performance.

Here is a sample SMFPRMxx member that will create interval records every 2 minutes, and record the following SMF record types:

- 30 - Address space
- 70-79 - RMF
- 82 - Crypto
- 88-90 - System Logger, Usage & System Data
- 101 - DB2
- 110 - CICS®
- 120 - WebSphere

```
ACTIVE                       /*ACTIVE SMF RECORDING*/
    DSNAME(&SYSNAME..MAN1, &SYSNAME..MAN2)       /*TWO MAN DATASETS */
    LISTDSN                      /* LIST DATA SET STATUS AT IPL*/
    NOPROMPT                     /* DON'T PROMPT THE OPERATOR  */
    INTVAL(02)                   /* SMF GLOBAL RECORDING INTERVAL */
    SYNCVAL(00)                  /* GLOBAL SYNC VALUE             */
    MAXDORM(3000)                /* WRITE AN IDLE BUFFER AFTER 30 MIN*/
    STATUS(010000)               /* WRITE SMF STATS AFTER 1 HOUR*/
    SID(&SYSNAME(1:4))           /* USE SYSNAME AS SID     */
    SUBSYS(STC,INTERVAL(SMF,SYNC),
                          TYPE(0,30,70:79,88:90,101,110,120))
```

Set the SMF recording interval to 2 minutes by using the 'SET SMF=xx' command to activate the SMFPRMxx member from SYSx.PARMLIB. Use the 'D SMF,O' command to display the parameters in effect.

Use a tool like WSWS to simulate an application stress load.

While the transactions are running, switch to SDSF and RMF to observe the transactions.

SMF record type 120: overview

Information resulting from the SMF data gathering process for WebSphere Application Server for z/OS is held in SMF record type 120.

This information is typically presented with the help of a SMF data viewing tool. This record format description is intended to enable your tool providers to design a SMF data viewing tool. Your system administrators will use a SMF data viewing tool with a description presented by your tool provider, since it requires them to make proper selections that limit the amount of presentation data. For example, they might want to view a specific time frame and only specific containers, classes, and methods. They may also occasionally need to refer to the record descriptions.

Two types of SMF records can be produced: *activity records* and *interval records*.

- Activity records are gathered as each activity within a server is completed. An activity is a logical unit of business function. It can be a server or user-initiated transaction.
- Interval records consist of data gathered at installation-specified intervals and provide capacity planning and reliability information.

Eight records can be produced:

- the Server Activity record: Subtype 1
- the Server Interval record: Subtype 3
- the J2EE Container Activity Record: Subtype 5

- the J2EE Container Interval Record: Subtype 6
- the WebContainer Activity record: Subtype 7
- the WebContainer Interval record: Subtype 8
- the Request Activity record: Subtype 9
- the Outbound Request record: Subtype 10

For additional information about using SMF records, see *z/OS MVS System Management Facilities (SMF), SA22-7630*.

Viewing records with the SMF Browser

You can use the SMF Browser, that is available on the product download site, to display record type 120. To download the SMF Browser, go to: https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=zosos390.

The documentation that is provided with the download package describes how to use this tool.

SMF record type 120 (78) - WebSphere Application Server performance statistics

WebSphere Application Server writes record type 120 to collect WebSphere Application Server performance statistics.

For more information about SMF record types, see *z/OS MVS System Management Facilities (SMF)*.

All subtypes of the record type 120 have the following format:
- Standard header section
- Individual header extension for subtype x
- Product section
- Subtype-specific sections listed below.

Record type 120 has the following subtypes:
- **Subtype 1: Server activity record**
  - **Server activity section** (one section per record):

    Contains information about each activity that occurred within one server.
  - **Communication session section** (zero, one, or multiple sections per record):

    Contains information about each communication session.
  - **JVM heap section** (zero, one, or multiple sections per record):

    Contains information about the heap in a server region.
- **Subtype 3: Server interval record**
  - **Server interval section** (one section per record):

    Contains aggregated information about all activities that occurred within the specified server interval.
  - **Server region section** (zero, one, or multiple sections per record):

    Contains information about server regions in the specified interval.
- **Subtype 5: J2EE container activity record**
  - **J2EE container activity section** (one section per record):

    Contains information about each activity that occurred within one J2EE container.
  - **Bean section** (multiple (0..n) sections per record):

    Contains information about all beans involved in this activity.
  - **Bean method section** (multiple (0..n) sections per bean section):

    Contains information about all methods of this bean involved in this activity.
- **Subtype 6: J2EE container interval record**
  - **J2EE container interval section** (one section per record):

Contains aggregated information about all activities that occurred within one J2EE container in the specified interval.

– **Bean section** (multiple (0..n) sections per record, see subtype 5):

Contains information about all beans involved in this activity in the specified interval.

– **Bean method section** (multiple (0..n) sections per bean section, see subtype 5):

Contains information about all methods of this class involved in this activity in the specified interval.

- **Subtype 7: WebContainer activity record (Version 2)**
  – **WebContainer activity section** (one section per record):

  Contains information about each activity that occurred within one WebContainer.

  – **HttpSessionManager activity section** (one section per record):

  Contains information about all sessions involved in this activity.

  – **WebApplication activity section** (multiple (0..n) sections per record):

  Contains information about all WebApplications involved in this activity.

    - **Servlet activity section** (multiple (0..n) sections per WebApplication section):

      Contains information about all Servlets involved in this activity.

- **Subtype 8: WebContainer interval record (Version 2)**
  – **WebContainer interval section** (one section per record):

  Contains information about each activity that occurred within one WebContainer in the specified interval.

  – **HttpSessionManager interval section** (one section per record):

  Contains information about all sessions involved in this activity in the specified interval.

  – **WebApplication interval section** (multiple (0..n) sections per record):

  Contains information about all WebApplications involved in this activity in the specified interval.

    - **Servlet interval section** (multiple (0..n) sections per WebApplication section):

      Contains information about all Servlets involved in this activity in the specified interval.

- **Subtype 9: Request Activity record**
  – **Platform neutral server information section** (one section per record):

  Contains information about each activity that occurred within one server.

  – **z/OS server information section** (one section per record):

  Contains information about each server servant in the specified server.

  – **Platform neutral request information section**

  (zero or one section per record):

  Contains information about each request that was received by one server.

  This section is not applicable for an asynchronous request.

  – **z/OS request information section**

  (zero or one section per record):

  Contains information about each request that was received by each server servant in the specified server.

  This section is not applicable for an asynchronous request.

  – **z/OS formatted timestamps section**

  (zero or one section per record):

  Contains the date and time information for all of the actions that each server servant in the specified server performed. These sections are optional.

  This section is not applicable for an asynchronous request.

  – **Network data for HTTP, SIP and IIOP transports section** (zero or one section per record):

  Contains information about either the HTTP, SIP or IIOP transports that are associated with one server. There is a separate network data section for the HTTP requests, the SIP requests, and the IIOP requests.

  This section is not applicable for an asynchronous request.

  – **Classification data section** (multiple (0..n) sections per record):

Contains the classification information for each HTTP, SIP, and IIOP request received by a server. There is a separate classification data section for each piece of classification information.

This section is not applicable for an asynchronous request.

- **Security data section** (multiple (0..n) sections per record):

  Contains the security information for each request received by a server. There is a separate security data section for each identity type. These sections are optional.

  This section is not applicable for an asynchronous request.
- **CPU usage breakdown section** (multiple (0..n) sections per record):

  Contains information about each item that was called and the CPU time that the task consumed, minus the time it spent waiting for tasks it initiated to complete. This calculation is different from the way CPU time is calculated in the container records. This section is optional.

  This section is optional for an asynchronous request.
- **User data section** (multiple (0..n) sections per record):

  Contains information that is added by applications that use the SMF 120 subtype 9 user data APIs.

  This section is optional for an asynchronous request.
- **Asynchronous data section** (Asynchronous data section (multiple (0..1) sections per record):

  Contains information that is created by requests that the server runs asynchronously.

  This section is always present for an asynchronous request.

- **Subtype 10: Outbound Request record**
  - **Platform neutral server information section** (one section per record):

    Contains information about the server that handled the outbound request.
  - **z/OS server information section** (one section per record):

    Contains information about the servant that handled the outbound request.
  - **Outbound Request information section** (one section per record):

    Contains general information about the outbound request.
  - **WOLA Outbound Request type specific section** (zero or one section per record):

    Contains WOLA specific information about the outbound request.
  - **Outbound Request transaction context section** (zero or one section per record):

    Contains the transactional information of the outbound request.
  - **Outbound Request security context section** (zero or one section per record):

    Contains the security information of the outbound request.
  - **Outbound Request CICS context section** (zero or one section per record):

    Contains the CICS context associated with the WOLA outbound request.
  - **OTMA Outbound Request type specific section** (zero or one section per record):

    Contains OTMA specific information about the outbound request.

Triplets

You can use triplets to build self-describing SMF records that contain various types of data sections and a varying number of each of these sections.

All data sections are described by triplets that consist of:

1. An offset that specifies the start position of the data
2. A length that describes the length of the section
3. A count that describes how many instances of the section are included in this record.

The two triplets that describe the product section and the general record information section (for example, the section describing the container itself in a container activity record) are located at fixed positions within the record. This allows one to start evaluating the record right after having evaluated the record header.

SMF record splitting

Since most of the WebSphere Application Server SMF records are used to describe variable-length data structures (for example, there might be hundreds of classes by container and hundreds of methods by class), the SMF records may be larger than the maximum record size supported by SMF (32KB). In this case, the logical records need to be split into several physical records.

Each of those physical records needs to be self-describing and self-contained. *Self-describing* indicates what we described in the paragraph on triplets before; it is a purely mechanical structure to help read a record. *Self-contained* indicates that, even if we have only a subset of the physical records at hand that together describe the original logical record, we need to be able to evaluate these records, combine the information stored in them, and set an 'incomplete' flag. This is required since, as we break up a logical record into physical records and write them to SMF one after the other, SMF might decide that only the first few physical records fit into the primary SMF dump dataset whereas the remaining physical records are written into an alternate SMF dump dataset. At the time when a formatted SMF dump dataset is evaluated, we may not assume that all physical records that make up one logical record are present. For example, self-containedness of a physical container activity record means that it contains the description of the container, but not necessarily all of its classes.

We use a similar splitting mechanism like the one that is currently used in the RMF product. Note that in the case of container records (subtypes 5, 6,7, and 8), we cannot assume that records will be split at a class boundary, but we must consider the case when the methods that belong to one class also need to be split over multiple physical records.

Note: The section length numbers used throughout the following diagrams are only for demonstrative purposes. In particular, the arrows indicating 32K boundaries or the total length of the records are placed at random. You can fit many more classes and methods into a physical record than suggested by the diagrams.

Record environment and mapping

This page provides information on record environments and record mapping.

Record environment

The following conditions exist for the generation of this record:

- **Macro** SMFWTM (record exit: IEFU83)
  Mode Task
  Addressing mode
  31-bit

Record mapping

For a description of the common SMF record header fields and the triplet fields (offset/length/number), if applicable, that locate the other sections on the record, see Header/Self-defining section.

For a description of triplets, see Using Triplets and MVS System Management Facilities (SMF)" (SA22-7630).

Header/self-defining section:

These tables describe the header/self-defining section of an SMF record.

Table 44. Header/self-defining section of an SMF record. This section contains the common SMF record headers fields and the triplet fields (offset/length/number), if applicable, that locate the other sections on the record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120LEN	2	binary	Record length. This field and the next field (total of four bytes) form the RDW (record descriptor word). See "Standard SMF record header" in MVSSystem Management Facilities (SMF)" (SA22-7630), for a detailed description.
2	2	SM120SEG	2	binary	Segment descriptor (see record length field)
4	4	SM120FLG	1	binary	**Bit meaning when set** 0: New SMF record format 1: Subtypes used 2: Reserved 3-6: Version indicators* 7: Reserved *See "Standard SMF record header" in MVSSystem Management Facilities (SMF)" (SA22-7630), for a detailed description.
5	5	SM120RTY	1	binary	Record type 120(X'78')
6	6	SM120TME	4	binary	Time since midnight, in hundredths of a second, that the record was moved into the SMF buffer.
10	A	SM120DTE	4	packed	Date when the record was moved into the SMF buffer, in the form 0*cyydddF*. See "Standard SMF record header" in MVSSystem Management Facilities (SMF)" (SA22-7630), for a detailed description.
14	E	SM120SID	4	EBCDIC	System identification (from the SMFPRMxx SID parameter)
18	12	SM120SSI	4	EBCDIC	Subsystem identification from SUBSYS parameter
22	16	SM120RST	2	binary	Record subtype: 1: Server activity 2: Container activity 3: Server interval 4: Container interval. 5: J2EE container activity 6: J2EE container interval 7: WebContainer activity 8: WebContainer interval 9: Request Activity record 10: Outbound Request record
24	18	SM120TRN	4	binary	Number of triplets in the record.
28	1C	SM120PRS	4	binary	Offset to product section from RDW.
32	20	SM120PRL	4	binary	Length of product section.
36	24	SM120PRN	4	binary	Number of product sections.
Individual header extension for subtype 1					
40	28	SM120SAS	4	binary	Offset to server activity section from RDW
44	2C	SM120SAL	4	binary	Length of server activity section
48	30	SM120SAN	4	binary	Number of server activity sections
52	34	SM120CSS	4	binary	Offset to communication session section from RDW
56	38	SM120CSL	4	binary	Length of communication session section

Table 44. Header/self-defining section of an SMF record (continued). This section contains the common SMF record headers fields and the triplet fields (offset/length/number), if applicable, that locate the other sections on the record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
60	3C	SM120CSN	4	binary	Number of communication session sections
64	40	SM120JHS	4	binary	Offset to JVM heap section from RDW
68	44	SM120JHL	4	binary	Length of JVM heap section
72	48	SM120JHN	4	binary	Number of jvm heap sections
Individual header extension for subtype 3					
40	28	SM120SIS	4	binary	Offset to server interval section from RDW
44	2C	SM120SIL	4	binary	Length of server interval section
48	30	SM120SIN	4	binary	Number of server interval sections
The following triplet appears 0-n times; once for each server region section.					
52	34	SM120SRS	4	binary	Offset to server region section from RDW
56	38	SM120SRL	4	binary	Length of server region section
60	3C	SM120SRN	4	binary	Number of server region sections
Individual header extension for subtype 5					
40	28	SM120JA1	4	binary	Offset to J2EE container activity section from RDW
44	2C	SM120JA2	4	binary	Length of J2EE container activity section
48	30	SM120JA3	4	binary	Number of J2EE container activity sections
The following triplet appears 0-n times; once for each bean section.					
52	34	SM120JAS	4	binary	Offset to bean section from RDW
56	38	SM120JAL	4	binary	Length of bean section
60	3C	SM120JAN	4	binary	Number of bean sections
Individual header extension for subtype 6					
40	28	SM120JI1	4	binary	Offset to J2EE container interval section from RDW
44	2C	SM120JI2	4	binary	Length of J2EE container interval section
48	30	SM120JI3	4	binary	Number of J2EE container interval sections
The following triplet appears 0-n times; once for each bean section.					
52	34	SM120JIS	4	binary	Offset to bean section from RDW
56	38	SM120JIL	4	binary	Length of bean section
60	3C	SM120JIN	4	binary	Number of bean sections
Individual header extension for subtype 7					
40	28	SM120WA1	4	binary	Offset to WebContainer activity section from RDW.
44	2C	SM120WA2	4	binary	Length of WebContainer activity section.
48	30	SM120WA3	4	binary	Number of WebContainer activity sections.
52	34	SM120WA4	4	binary	Offset to HttpSessionManager activity section from RDW.
56	38	SM120WA5	4	binary	Length of HttpSessionManager activity section.
60	3C	SM120WA6	4	binary	Number of HttpSessionManager activity sections.
The following triplet appears 0-n times; once for each WebApplication section.					
64	40	SM120WA7	4	binary	Offset to WebApplication section from RDW.
68	44	SM120WA8	4	binary	Length of WebApplication section.

Table 44. Header/self-defining section of an SMF record (continued). This section contains the common SMF record headers fields and the triplet fields (offset/length/number), if applicable, that locate the other sections on the record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
72	48	SM120WA9	4	binary	Number of WebApplication sections.
Individual header extension for subtype 8					
40	28	SM120WI1	4	binary	Offset to WebContainer interval section from RDW.
44	2C	SM120WI2	4	binary	Length of WebContainer interval section.
48	30	SM120WI3	4	binary	Number of WebContainer interval sections.
52	34	SM120WI4	4	binary	Offset to HttpSessionManager interval section from RDW.
56	38	SM120WI5	4	binary	Length of HttpSessionManager interval section.
60	3C	SM120WI6	4	binary	Number of HttpSessionManager interval sections.
The following triplet appears 0-n times; once for each WebApplication section.					
64	40	SM120WI7	4	binary	Offset to WebApplication section from RDW.
68	44	SM120WI8	4	binary	Length of WebApplication section.
72	48	SM120WI9	4	binary	Number of WebApplication sections.
Individual header extension for subtype 9					
24	18	SM1209AA	4	binary	Subtype version number
28	1C	SM1209AB	4	binary	Number of triplets
32	20	SM1209AC	4	binary	Index of this record
36	24	SM1209AD	4	binary	Total number of records
40	28	SM1209AE	8	EBCDIC	Record continuation token
The following triplet appears for the Platform neutral server information section.					
48	30	SM1209AF	4	binary	The offset to the Platform neutral server information section
52	34	SM1209AG	4	binary	The length of the Platform neutral server information section
56	38	SM1209AH	4	binary	The number of Platform neutral server information sections
The following triplet appears for the z/OSserver information section.					
60	3C	SM1209AI	4	binary	The offset to the z/OSserver information section
64	40	SM1209AJ	4	binary	The length of the z/OSserver information section
68	44	SM1209AK	4	binary	The number of z/OSserver information sections
The following triplet appears for the Platform neutral request information section.					
72	48	SM1209AL	4	binary	The offset to the Platform neutral request information section
76	4C	SM1209AM	4	binary	The length of the Platform neutral request information section
80	50	SM1209AN	4	binary	The number of Platform neutral request information sections
The following triplet appears for the z/OSrequest information section.					
84	54	SM1209AO	4	binary	The offset to the z/OSrequest information section
88	58	SM1209AP	4	binary	The length of the z/OSrequest information section
92	5C	SM1209AQ	4	binary	The number of z/OSrequest information sections
The following triplet appears for the z/OS formatted timestamps section. This section contains zeros when the formatted timestamps are not being collected.					
96	60	SM1209AR	4	binary	The offset to the z/OSformatted timestamps section
100	64	SM1209AS	4	binary	The length of the z/OSformatted timestamps section
104	68	SM1209AT	4	binary	The number of z/OSformatted timestamps sections

Table 44. Header/self-defining section of an SMF record (continued). This section contains the common SMF record headers fields and the triplet fields (offset/length/number), if applicable, that locate the other sections on the record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
The following triplet contains network information. Only one version of this section appears for IIOP, HTTP transports, and SIP transports. This section does not appear for MDBs, or for internal protocols.					
108	6C	SM1209AU	4	binary	The offset to the Network data for HTTP, SIP and IIOP transports section
112	70	SM1209AV	4	binary	The length of the Network data for HTTP, SIP and IIOP transports section
116	74	SM1209AW	4	binary	The number of Network data for HTTP, SIP and IIOP transports sections
The following triplet appears for the Classification data section.					
120	78	SM1209AX	4	binary	The offset to the Classification data section
124	7C	SM1209AY	4	binary	The length of the Classification data section
128	80	SM1209AZ	4	binary	The number of Classification data sections
The following triplet appears for the Security data section.					
132	84	SM1209BA	4	binary	The offset to the Security data section
136	88	SM1209BB	4	binary	The length of the Security data section
140	8C	SM1209BC	4	binary	The number of Security data sections
The following triplet appears 0-30 times for the CPU usage breakdown sections.					
144	90	SM1209BD	4	binary	The offset to the CPU usage breakdown
148	94	SM1209BE	4	binary	The length of the CPU usage breakdown section
152	98	SM1209BF	4	binary	The number of CPU usage breakdown sections
The following triplet appears for the user data section.					
156	9C	SM1209FB	4	binary	The offset to the user data section
160	A0	SM1209FC	4	binary	The length of the user data section
164	A4	SM1209FD	4	binary	The number of user data sections
The following triplet appears for the asynchronous data section.					
168	A8	SM1209GB	4	binary	The offset to the asynchronous data section
172	AC	SM1209GC	4	binary	The length of the asynchronous data section
176	B0	SM1209GD	4	binary	The number of asynchronous data sections
180	B4	*	24		Reserved
Individual header extension for subtype 10					
24	18	SM120AAA	4	binary	Subtype version number
28	1C	SM120AAB	4	binary	Number of triplets
32	20	SM120AAC	4	binary	Index of this record
36	24	SM120AAD	4	binary	Total number of records
40	28	SM120AAE	8	EBCDIC	Record continuation token
The following triplet appears for the Platform neutral server information section.					
48	30	SM120AAF	4	binary	The offset to the Platform neutral server information section
52	34	SM120AAG	4	binary	The length of the Platform neutral server information section
56	38	SM120AAH	4	binary	The number of Platform neutral server information sections
The following triplet appears for the z/OSserver information section.					
60	3C	SM120AAI	4	binary	The offset to the z/OSserver information section

Table 44. Header/self-defining section of an SMF record (continued). This section contains the common SMF record headers fields and the triplet fields (offset/length/number), if applicable, that locate the other sections on the record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
64	40	SM120AAJ	4	binary	The length of the z/OSserver information section
68	44	SM120AAK	4	binary	The number of z/OSserver information sections
The following triplet appears for the common outbound request information section.					
72	48	SM120AAL	4	binary	The offset to the Outbound request information section
76	4C	SM120AAM	4	binary	The length of the Outbound request information section
80	50	SM120AAN	4	binary	The number of Outbound request information sections
The following triplet appears for the WOLA outbound request type specific information section.					
84	54	SM120AAR	4	binary	The offset to the WOLA outbound request type specific information section
88	58	SM120AAS	4	binary	The length of the WOLA outbound request type specific information section
92	5C	SM120AAT	4	binary	The number of WOLA outbound request type specific information sections
The following triplet appears for the outbound request transaction context section.					
96	60	SM120AAU	4	binary	The offset to the outbound request transaction context section
100	64	SM120AAV	4	binary	The length of the outbound request transaction context section
104	68	SM120AAW	4	binary	The number of outbound request transaction context sections
The following triplet appears for the outbound request security context section.					
108	6C	SM120AAX	4	binary	The offset to the outbound request security context section.
112	70	SM120AAY	4	binary	The length of the outbound request security context section.
116	74	SM120AAZ	4	binary	The number of outbound request security context sections
The following triplet appears for the outbound request CICS context section.					
120	78	SM120AA1	4	binary	The offset to the outbound request CICS context section
124	7C	SM120AA2	4	binary	The length of the outbound request CICS context section
128	80	SM120AA3	4	binary	The number of outbound request CICS context sections
The following triplet appears for the OTMA outbound request type specific section.					
132	84	SM120AA4	4	binary	The offset to the OTMA outbound request type specific section
136	88	SM120AA5	4	binary	The length of the OTMA outbound request type specific section
140	8C	SM120AA6	4	binary	The number of OTMA outbound request type specific sections
The following area is reserved for future triplets.					
144	90	*	60	binary	Reserved

Product section:

These tables describe the product section of an SMF record.

Product section

Table 45. Product section. This section contains the product that generated the record.

Offset	Offset	Name	Length	Format	Description
0	0	SM120MFV	4	binary	CB SMF version

Table 45. Product section (continued). This section contains the product that generated the record.

Offset	Offset	Name	Length	Format	Description
4	4	SM120COD	8	EBCDIC	Character codeset in which strings in the SMF record are encoded
12	C	SM120END	4	binary	Encode of numbers in the SMF record
16	10	SM120TSF	4	binary	Encoding of timestamps: 1: S390STCK64: The time values are encoded in 64-bit S/390Store Clock format.
Reassembly information.					
20	14	SM120IXR	4	binary	Index of this record
24	18	SM120NRC	4	binary	Total number of records
28	1C	SM120NTR	4	binary	Total number of triplets

SMF Subtype 1: Server activity record:

The server activity SMF record is used to record activity that is running inside a WebSphereApplication Server for z/OS. This record can be used to perform basic charge-back accounting and to profile your applications to determine, in detail, what is happening inside the WebSphereApplication Server transaction server.

A single record is created for each activity that is run inside a server or server instance. If the activity runs in multiple servers, then a record is written for each server.

You can activate this record through the administrative console by setting **server_SMF_server_activity_enabled=1 (or server_SMF_server_activity_enabled=true)**. See “Using the administrative console to enable properties for specific SMF record types” on page 209for instructions.

Server activity record schema

This section includes Subtype 1: Server activity record.

Server activity section

Table 46. Server activity section. The Server activity section contains information about each activity that occurred within one server.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120HNM	64	EBCDIC	WebSphereApplication Server for z/OStransaction server host name
64	40	SM120SNA	8	EBCDIC	WebSphereApplication Server for z/OStransaction server name
72	48	SM120INA	8	EBCDIC	WebSphereApplication Server for z/OStransaction server instance name
80	50	SM120SNM	4	binary	Total number of server servants that were involved to process this activity. If applicable, up to the first five server servant address space IDs are listed within the next five fields.
84	54	SM120SR1	4	binary	The specific WebSphereApplication Server for z/OStransaction server instance server servant where the request ran
88	58	SM120SR2	4	binary	The specific WebSphereApplication Server for z/OStransaction server instance server servant where the request ran

Table 46. Server activity section (continued). The Server activity section contains information about each activity that occurred within one server.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
92	5C	SM120SR3	4	binary	The specific WebSphereApplication Server for z/OStransaction server instance server servant where the request ran
96	60	SM120SR4	4	binary	The specific WebSphereApplication Server for z/OStransaction server instance server servant where the request ran
100	64	SM120SR5	4	binary	The specific WebSphereApplication Server for z/OStransaction server instance server servant where the request ran
104	68	SM120CRE	8	EBCDIC	The user credentials under which the activity began. Due to deferred security authentication, the user credentials assigned to the request when it first reaches the server will often be the unauthenticated guest ID, and not the ID of the authenticated user that submitted the request.
112	70	SM120ATY	4	binary	Type of activity that this record references: 1: Method request: This record refers to a method request that is not part of a global transaction. 2: Transaction: This record refers to a transaction.
116	74	SM120AID	20	HEX	Identity of the activity
136	88	SM120WLM	8	HEX	WLM enclave token
144	90	SM120AST	16	S390STCK	Activity start time
160	A0	SM120AET	16	S390STCK	Activity stop time
176	B0	SM120NIM	4	binary	Number of input methods
180	B4	SM120NGT	4	binary	Number of global transactions that were started in the server servant
184	B8	SM120NLT	4	binary	Number of local transactions that were started in the server servant
188	BC	SM120J2E	4	binary	J2EE server
192	C0	SM120CEL	8	EBCDIC	WebSphereApplication Server for z/OScell name
200	C8	SM120NOD	8	EBCDIC	WebSphereApplication Server for z/OSnode name
208	D0	SM120WCP	8	binary	Total CPU time accumulated by the WLM enclave. TOD clock format (bit 51 = microseconds).

Communications session section

Table 47. Communications session section. There are zero, one, or multiple sections per record. The Communications session section contains information about each communication session.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120CSH	8	HEX	Communications session handle
8	8	SM120CSA	64	EBCDIC	Communications session address

Table 47. Communications session section (continued). There are zero, one, or multiple sections per record. The Communications session section contains information about each communication session.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
72	48	SM120CSO	4	binary	Communications session optimization 1: Local communications session: The session is a local OS/390®optimized communications session. 2: Remote communications session: The session is a remote communications session. 3: Remote encrypted (SSL) 4: Remote within sysplex. 5: HTTP session. 6: HTTP encrypted session. 7. Message-driven bean session
76	4C	SM120SDR	4	binary	Data received; the number of bytes received by the server. 'FFFFFFFF'X indicates the 4-byte field is too small. Use SM120CDR, an 8-byte field, instead.
80	50	SM120SDT	4	binary	Data transferred; the number of bytes transferred from the server back to the client. 'FFFFFFFF'X indicates the 4-byte field is too small. Use SM120CDT, an 8-byte field, instead.
84	54	SM120CDR	8	binary	Data received; the number of bytes received by the server.
92	5C	SM120CDT	8	binary	Data transferred; the number of bytes transferred from the server back to the client.

JVM Heap section

There are zero, one, or multiple sections per record. The JVM heap section contains information about the heap in each server servant.

Table 48. JVM Heap section. The information in the JVM heap section comes from the QueryGCStatus() JNI function.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120JHA	4	binary	Servant address space ID
4	4	SM120JHH	4	binary	The heap for which the following data applies.
8	8	SM120JHC	4	binary	The total number of allocation failures on this heap or, if querying shared storage, the subpool identifier. A negative value indicates the information is for the shared memory page pool.
12	C	SM120JHF	8	binary	The total number of free bytes in the heap/subpool/page pool.
20	14	SM120JHT	8	binary	The total number of bytes in the heap, subpool, or page pool.

SMF Subtype 3: Server interval record:

The purpose of the server interval SMF record is to record activity that is running inside a WebSphereApplication Server for z/OS. This record is produced at regular intervals and is an aggregate of the work that ran inside the server instance during the interval.

A single record is created for each server instance that has interval recording active during the interval. When a server is configured with multiple server instances, each server instance writes a record and the records from all the server instances must be merged by whoever is looking at the SMF records to get a complete view of the work that ran inside the logical server.

You can activate this record through the administrative console by setting **server_SMF_server_interval_enabled=1 (or server_SMF_server_interval_enabled=true)**. You can specify an interval through the administrative console by setting server_SMF_interval_length=n, where n is the desired number of seconds.

Server interval record schema

This section includes Subtype 3: Server interval record.

Server interval section

Table 49. Server interval section. The server interval section contains information about each activity that occurred within one server.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120HN2	64	EBCDIC	WebSphereApplication Server for z/OStransaction server host name
64	40	SM120SNI	8	EBCDIC	WebSphereApplication Server for z/OStransaction server name
72	48	SM120INI	8	EBCDIC	WebSphereApplication Server for z/OStransaction server instance name
80	50	SM120SST	16	S390STCK	Time that the sample began in the server
96	60	SM120SET	16	S390STCK	Time that the sample ended
112	70	SM120NG2	4	binary	Number of global transactions that have run through the server instance during the interval that have been initiated by the server instance during the interval
116	74	SM120NL2	4	binary	Number of local transactions that have been initiated by the server instance during the interval
120	78	SM120NCS	4	binary	Reserved
124	7C	SM120NCA	4	binary	The number of communications sessions that have been active during the interval
128	80	SM120NLS	4	binary	Reserved
132	84	SM120NLA	4	binary	Number of active local communication sessions that have been attached and active within the server instance during the interval
136	88	SM120NRS	4	binary	Reserved
140	8C	SM120NRA	4	binary	Number of active remote communication sessions that have been attached and active within the server instance during the interval
144	90	SM120BTS	4	binary	Number of bytes that have been transferred to the server from all attached clients 'FFFFFFFF'X indicates the 4-byte field is too small. Use SM120ITS, an 8-byte field, instead.
148	94	SM120BFS	4	binary	Number of bytes that have been sent from the server to all attached clients 'FFFFFFFF'X indicates the 4-byte field is too small. Use SM120IFS, an 8-byte field, instead.

Table 49. Server interval section (continued). The server interval section contains information about each activity that occurred within one server.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
152	98	SM120BTL	4	binary	Number of bytes that have been transferred to the server from all locally attached clients 'FFFFFFFF'X indicates the 4-byte field is too small. Use SM120ITL, an 8-byte field, instead.
156	9C	SM120BFL	4	binary	Number of bytes that have been transferred from the server to all locally attached clients 'FFFFFFFF'X indicates the 4-byte field is too small. Use SM120IFL, an 8-byte field, instead.
160	A0	SM120BTR	4	binary	Number of bytes that have been transferred to the server from all remotely attached clients 'FFFFFFFF'X indicates the 4-byte field is too small. Use SM120ITR, an 8-byte field, instead.
164	A4	SM120BFR	4	binary	Number of bytes that have been transferred from the server to all remotely attached clients 'FFFFFFFF'X indicates the 4-byte field is too small. Use SM120IFR, an 8-byte field, instead.
168	A8	SM120J2	4	binary	J2EE server.
172	AC	SM120CL1	8	EBCDIC	WebSphereApplication Server for z/OStransaction server cell name
180	B4	SM120ND1	8	EBCDIC	WebSphereApplication Server for z/OStransaction server node name
188	BC	SM120NHS	4	binary	Reserved
192	C0	SM120NHA	4	binary	Number of HTTP communication sessions that have been attached and active within the server instance during the interval
196	C4	SM120BTH	4	binary	Number of bytes that have been transferred to the server from all HTTP attached clients 'FFFFFFFF'X indicates the 4-byte field is too small. Use SM120ITH, an 8-byte field, instead.
200	C8	SM120BFH	4	binary	Number of bytes that have been transferred from the server to all HTTP attached clients 'FFFFFFFF'X indicates the 4-byte field is too small. Use SM120IFH, an 8-byte field, instead.
204	CC	SM120TEC	8	binary	Total CPU time accumulated by the WLM enclaves. TOD clock format (bit 51 = microseconds).
212	D4	SM120ITS	8	binary	Number of bytes that have been transferred to the server from all attached clients.
220	DC	SM120IFS	8	binary	Number of bytes that have been sent from the server to all attached clients
228	E4	SM120ITL	8	binary	Number of bytes that have been transferred to the server from all locally attached clients
236	EC	SM120IFL	8	binary	Number of bytes that have been transferred from the server to all locally attached clients
244	F4	SM120ITR	8	binary	Number of bytes that have been transferred to the server from all remotely attached clients
252	FC	SM120IFR	8	binary	Number of bytes that have been transferred from the server to all remotely attached clients

Table 49. Server interval section (continued). The server interval section contains information about each activity that occurred within one server.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
260	104	SM120ITH	8	binary	Number of bytes that have been transferred to the server from all HTTP attached clients
268	10C	SM120IFH	8	binary	Number of bytes that have been transferred from the server to all HTTP attached clients
276	114	SM120ITP	8	binary	Number of bytes that have been transferred to the server from all SIP attached clients
284	11C	SM120IFP	8	binary	Number of bytes that have been transferred from the server to all SIP attached clients.
292	124	SM120NPA	4	Binary	Number of SIP communication sessions that have been attached and active within the server instance during the interval
296	128	SM120BTP	4	binary	Number of bytes that have been transferred to the server from all SIP attached clients. 'FFFFFFFF"X indicates the 4-byte field is too small. Use SM120ITP, an 8-byte field, instead.
300	12C	SM120BFP	4	binary	Number of bytes that have been transferred from the server to all SIP attached clients. 'FFFFFFFF"X indicates the 4-byte field is too small. Use SM120IFP, an 8-byte field, instead.
304	130	SM120IR1	4		Reserved

Server servant section

Table 50. Server servant section. There are zero, one, or multiple sections per record. The server servant section contains information about each server servant in the specified server interval.

Offset	Offset	Name	Length	Format	Description
0	0	SM120SSA	4	binary	Servant address space ID
4	4	SM120SNT	4	binary	Number of triplets.
The following triplet appears 0-n times; once for each heap id section.					
8	8	SM120SSO	4	binary	Offset to heap id section from the beginning of this server servant section.
12	C	SM120SSL	4	binary	Length of heap id section.
16	10	SM120SSN	4	binary	Number of heap id sections.

Subtype 3: Heap ID section

Table 51. Subtype 3: Heap ID section. There are multiple (0..n) sections per server servant section. The Heap id section contains information about all heaps of this server servant involved in this activity.

Offset	Offset	Name	Length	Format	Description
0	0	SM120HIH	4	binary	The heap for which the following data applies.
4	4	SM120HIC	4	binary	Number of allocation failures on this heap during the interval.
8	8	SM120HI1	8	binary	Minimum number of bytes during the interval.
16	10	SM120HI2	8	binary	Maximum number of bytes during the interval.
24	18	SM120HI3	8	binary	Average number of bytes during the interval.
32	20	SM120HI4	8	binary	Minimum number of free bytes during the interval.

Table 51. Subtype 3: Heap ID section (continued). There are multiple (0..n) sections per server servant section. The Heap id section contains information about all heaps of this server servant involved in this activity.

Offset	Offset	Name	Length	Format	Description
40	28	SM120HI5	8	binary	Maximum number of free bytes during the interval.
48	30	SM120HI6	8	binary	Average number of free bytes during the interval.

SMF Subtype 5: J2EE container activity record (Version 2):

The purpose of the J2EE container activity SMF record is to record activity within a J2EE container that is located inside the WebSphereApplication Server transaction server.

This record can be used to perform basic charge-back accounting, application profiling, problem determination, and capacity planning. A single record is created for each activity that is run within a J2EE container located inside a WebSphereApplication Server transaction server.

You can activate this record through the administrative console by setting **server_SMF_container_activity_enabled=1 (or server_SMF_container_activity_enabled=true)**.

J2EE container activity record (Version 2) schema

This section includes Subtype 5: J2EE container activity record (Version 2).

J2EE container activity section

Table 52. J2EE container activity section. There is one section per record. The J2EE container activity section contains information about each activity that occurred within one J2EE container.

Offset	Offset	Name	Length	Format	Description
0	0	SM120JA4	64	EBCDIC	WebSphereApplication Server for z/OStransaction server host name
64	40	SM120JA5	8	EBCDIC	WebSphereApplication Server for z/OStransaction server name
72	48	SM120JA6	8	EBCDIC	WebSphereApplication Server for z/OStransaction server instance name
80	50	SM120JA7	4	binary	The specific WebSphereApplication Server for z/OStransaction server instance server servant where the request ran
84	54	SM120JA8	512	Unicode	WebSphereApplication Server for z/OScontainer name.
596	254	SM120JA9	8	HEX	The WLM enclave token
604	25C	SM120JAA	4	binary	RESERVED
608	260	SM120JAB	20	HEX	The identity of the activity
628	274	SM120CL2	8	EBCDIC	Cell
636	27C	SM120ND2	8	EBCDIC	Node

Bean section

Table 53. Bean section. There are multiple sections per record. The bean section contains information about all beans involved in this activity.

Offset	Offset	Name	Length	Format	Description
0	0	SM120JB1	512	Unicode	AMCName of the bean activated by the container. **Note:** If the length of the AMCName exceeds 256 DBCS characters (512 bytes), the rightmost 256 characters are recorded.
512	200	SM120JB2	60	binary	UUID based AMC name
572	23C	SM120JB3	4	binary	The bean's type. 2: Stateless session bean. 3: Stateful session bean. 4: BMP entity bean. 5: CMP entity bean. 6. Message-driven bean.
576	240	SM120JB4	4	binary	RESERVED
580	244	SM120JB5	4	binary	RESERVED
584	248	SM120JB6	4	binary	RESERVED
588	24C	SM120JB7	4	binary	The bean's reentrance policy. 0: Not reentrant within transaction. 1: Reentrant within transaction.
592	250	SM120JB8	4	binary	RESERVED
596	254	SM120JMC	4	binary	RESERVED
600	258	SM120JM6	4	binary	RESERVED
604	25C	SM120JB9	4	binary	Number of method triplets in this bean section
The following triplet appears 0-n times; once for each bean method section.					
608	260	SM120JBS	4	binary	Offset to bean method section from the beginning of this bean section
612	264	SM120JBL	4	binary	Length of bean method section
616	268	SM120JBN	4	binary	Number of bean method sections

Bean method section

Table 54. Bean method section. There are multiple sections per bean section. The bean method section contains information about all methods of beans involved in this activity.

Offset	Offset	Name	Length	Format	Description
0	0	SM120JM1	1,024	Unicode	The name of the method including its signature in its externalized, human-readable form. If the length of the method exceeds 512 DBCS characters (1024 bytes), the leftmost 512 characters are recorded.
1024	400®	SM120JM2	4	binary	The number of times the method was invoked during the activity.
1028	404	SM120JM3	4	binary	Average response time. The response time is measured in milliseconds (the granularity provided by the JVM - hopefully, it will be equal to 0 in most cases).
1032	408	SM120JM4	4	binary	Maximum response time. The response time is measured in milliseconds.

Table 54. Bean method section (continued). There are multiple sections per bean section. The bean method section contains information about all methods of beans involved in this activity.

Offset	Offset	Name	Length	Format	Description
1036	40C	SM120JM5	4	binary	The bean method's transaction policy. Values from com.ibm.websphere.csi. TransactionAttribute.java: 0: "TX_NOT_SUPPORTED" 1: "TX_BEAN_MANAGED" 2: "TX_REQUIRED" 3: "TX_SUPPORTS" 4: "TX_REQUIRES_NEW" 5: "TX_MANDATORY" 6: "TX_NEVER"
1040	410	SM120JM8	4	binary	RESERVED.
1044	414	SM120JM9	4	binary	RESERVED.
1048	418	SM120JMA	512	Unicode	List of ejbRoles associated with the method. Separator character: ";" (semicolon). If the length of the concatenated string exceeds 256 characters (512 bytes), only its leftmost 256 characters are recorded.
1560	618	SM120JMB	4	binary	RESERVED.
1564	61C	SM120JMD	4	binary	RESERVED.
1568	620	SM120JME	4	binary	ejbLoad: # of invocations
1572	624	SM120JMF	4	binary	ejbLoad: avg execution time
1576	628	SM120JMG	4	binary	ejbLoad: max execution time
1580	62C	SM120JMH	4	binary	ejbStore: # of invocations
1584	630	SM120JMI	4	binary	ejbStore: avg execution time
1588	634	SM120JMJ	4	binary	ejbStore: max execution time
1592	638	SM120JMK	4	binary	ejbActivate: # of invocations
1596	63C	SM120JML	4	binary	ejbActivate: avg execution time
1600	640	SM120JMM	4	binary	ejbActivate: max execution time
1604	644	SM120JMN	4	binary	ejbPassivate: # of invocations
1608	648	SM120JMO	4	binary	ejbPassivate: avg execution time
1612	64C	SM120JMP	4	binary	ejbPassivate: max execution time
1616	650	SM120JMQ	8	binary	Average cpu time in microseconds.
1624	658	SM120JMR	8	binary	Minimum cpu time in microseconds.
1632	660	SM120JMS	8	binary	Maximum cpu time in microseconds.

SMF Subtype 6: J2EE container interval record (Version 2):

The purpose of the J2EE container interval SMF record is to record activity within a J2EE container that is located inside the WebSphereApplication Server transaction server.

This record is produced at regular intervals and is an aggregate of the activities running inside a J2EE container during the interval. This record can be used to perform application profiling, problem determination, and capacity planning.

A single record is created for each active J2EE container located in a WebSphereApplication Server transaction server within the interval being recorded. If there is more than one server instance associated with a server, a record for the container will exist for each server instance. To get a common view of the work running in the J2EE container during the interval, you must merge the records after processing.

You can specify an interval through the WebSphereApplication Server administrative console by setting **server_SMF_interval_length=n**, where n is the desired number of seconds.

You can activate this record by setting **server_SMF_container_interval_enabled=1 (or server_SMF_container_interval_enabled=true)** on the administrative console.

J2EE container interval record (Version 2) schema

This section includes Subtype 6: J2EE container interval record (Version 2).

J2EE container interval section

Table 55. J2EE container interval section. There is one section per record. The J2EE container interval section contains information about each activity that occurred within one J2EE container in the specified interval.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120JI4	64	EBCDIC	The WebSphereApplication Server for z/OStransaction server host name.
64	40	SM120JI5	8	EBCDIC	The WebSphereApplication Server for z/OStransaction server name.
72	48	SM120JI6	8	EBCDIC	The WebSphereApplication Server for z/OStransaction server instance name.
80	50	SM120JI7	512	Unicode	The WebSphereApplication Server for z/OScontainer name. **Note:** This is hardcoded to "Default" for the 4.0.1 time frame.
592	250	SM120JI8	16	S390STCK	The time that the sample began in the server.
608	260	SM120JI9	16	S390STCK	The time that the sample ended.
624	270	SM120CL3	8	EBCDIC	Cell
632	278	SM120ND3	8	EBCDIC	Node

Subtype 6: Bean section:

See Subtype 5: Bean section

Subtype 6: Bean method section:

See Subtype 5: Bean method section

SMF Subtype 7: WebContainer activity record (Version 2):

The purpose of the WebContainer activity SMF record is to record activity within a WebContainer running inside a WebSphereApplication Server for z/OStransaction server.

The web container is deployed within an EJB and runs within the EJB container. The WebContainer acts as a web server handling HTTP sessions and servlets. The EJB container is not aware of the work the WebContainer does. Instead, the EJB container only records that the EJB has been dispatched. Meanwhile, the WebContainer gathers the detailed information, such as HTTP sessions, servlets, and their respective performance data. A single WebContainer Activity record is created for each activity that is run within a web container.

WebContainer SMF recording is activated and deactivated along with the activation and deactivation of SMF recording for the J2EE container.

WebContainer activity record (Version 2) schema

This section includes Subtype 7: WebContainer activity record (Version 2).

WebContainer activity section

.

Table 56. WebContainer activity section. There is one section per record. The WebContainer activity section contains information about each activity that occurred within one web container.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120WAA	64	EBCDIC	The WebSpheretransaction server host name.
64	40	SM120WAB	8	EBCDIC	The WebSpheretransaction server name.
72	48	SM120WAC	8	EBCDIC	The WebSpheretransaction server instance name.
80	50	SM120WAD	8	HEX	The WLM enclave token.
88	58	SM120WAE	20	HEX	The identity of the activity.
108	6C	SM120WAF	16	S390STCK	The time the activity began in the server.
124	7C	SM120WAG	16	S390STCK	The time the activity ended.
140	8C	SM120CL4	8	EBCDIC	Cell
148	94	SM120ND4	8	EBCDIC	Node

HttpSessionManager section

Table 57. HttpSessionManager section. There is one section per record. The HttpSessionManager section contains information about all (there may be zero or one) http sessions associated to one single activity.

Offset	Offset	Name	Length	Format	Description
0	0	SM120WAH	4	binary	"created Sessions": Number of http sessions that were created.
4	4	SM120WAI	4	binary	"invalidatedSessions": Number of http session that were invalidated.
8	8	SM120WAJ	4	binary	"activeSessions": Number of http sessions that were referenced during this activity.
12	C	SM120WAK	4	binary	"sessionLifeTime": lifetime of the session in milliseconds. If "invalidatedSessions" > 0, this is the average lifetime (in milliseconds) of the invalidated http session.

WebApplication section

Table 58. WebApplication section. There are multiple (0-n) sections per record. The WebApplication section contains information about all WebApplications involved in this activity.

Offset	Offset	Name	Length	Format	Description
0	0	SM120WAL	256	Unicode	The name of the WebApplication.
256	100	SM120WAM	4	binary	Number of servlet triplets in this web application section.
The following triplet appears 0-n times, once for each servlet section.					
260	104	SM120WAN	4	binary	Offset to servlet section from the beginning of this WebApplication section.
264	108	SM120WAO	4	binary	Length of servlet section.

Table 58. WebApplication section (continued). There are multiple (0-n) sections per record. The WebApplication section contains information about all WebApplications involved in this activity.

Offset	Offset	Name	Length	Format	Description
268	10C	SM120WAP	4	binary	Number of servlet sections.

Servlet activity section

Table 59. Servlet activity section. There are multiple (0-n) sections per WebApplication section. The Servlet activity section contains information about each servlet associated with WebApplications involved in this activity.

Offset	Offset	Name	Length	Format	Description
0	0	SM120WAQ	256	Unicode	The name of the servlet.
256	100	SM120WAR	4	binary	"responseTime": Response time in milliseconds.
260	104	SM120WAS	4	binary	"numErrors": The number of errors that were encountered during the servlet execution.
264	108	SM120WAT	4	binary	"loaded": 0: The servlet did not have to be loaded as a result of this request. 1: The servlet had to be loaded as the result of this request.

268	10C	SM120WAU	16	EBCDIC	"loadedSince": Timestamp from System.currentTimeMillis() when the servlet was loaded, in HEX format.
					
					Sample: The data as it appears in the record has the format
					`e7ef7c577c`
					
					, which needs to be converted to a Javalong:
					`996155348860`
					
					. The Javalong digits can be converted to java.util.Date:
					`Thu Jul 26 15:49:08 GMT+02:00 2001`

284	11C	SM120CPU	8	binary	Cpu time in microseconds.

SMF Subtype 8: WebContainer interval record (Version 2):

The purpose of the WebContainer interval SMF record is to record activity within a WebContainer running inside a WebSphereApplication Server for z/OStransaction server.

The web container acts as a web server handling HttpSessions and Servlets. The EJB container is not aware of the purpose of the WebContainer activity record and only records that the EJB has been dispatched, but does not gather any of the detailed information, such as HttpSessions, Servlets, and their respective performance data. A single WebContainer record is created for each web container.

In addition to data that is associated with an individual activity, there are some cases of web container work that are performed outside the scope of an individual request. For example, some instances of http session finalization and http session invalidation are performed asynchronously. In such a case a WebContainer interval record would record this data

WebContainer SMF recording is activated and deactivated along with the activation and deactivation of SMF recording for the J2EE container.

WebContainer interval record (Version 2) schema

WebContainer interval record (Version 2) schema.

WebContainer interval section

Table 60. WebContainer interval section. There is one section per record. The WebContainer interval section contains information about each activity that occurred within one WebContainer record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120WIA	64	EBCDIC	The WebSpheretransaction server host name.
64	40	SM120WIB	8	EBCDIC	The WebSpheretransaction server name.
72	48	SM120WIC	8	EBCDIC	The WebSpheretransaction server instance name.
80	50	SM120WID	16	S390STCK	The time the sample began.
96	60	SM120WIE	16	S390STCK	The time the sample ended.
112	70	SM120CL5	8	EBCDIC	Cell
120	78	SM120ND5	8	EBCDIC	Node

HttpSessionManager section

Table 61. HttpSessionManager section. There is one section per record. The HttpSessionManager section contains information about all (there may be zero or one) http sessions associated to one single activity.

Offset	Offset	Name	Length	Format	Description
0	0	SM120WIF	4	binary	"createdSessions": Number of http sessions that were created.
4	4	SM120WIG	4	binary	"invalidatedSessions": Number of http sessions that were invalidated.
8	8	SM120WIH	4	binary	"activeSessions": Current®number of http sessions that are actively referenced in the server at the end of the interval.
12	C	SM120WII	4	binary	"minActiveSessions": Minimum number of active http sessions during the interval..
16	10	SM120WIJ	4	binary	"maxActiveSessions": Maximum number of active http sessions during the interval.
20	14	SM120WIK	4	binary	"sessionLifeTime": Average lifetime (in milliseconds) of invalidated http sessions.
24	18	SM120WIL	4	binary	"sessionInvalidateTime": Average time (in milliseconds) that was required to process the invalidation of http sessions.
28	1C	SM120WIM	4	binary	"finalizedSessions": Number of sessions that were finalized.
32	20	SM120WIN	4	binary	"liveSessions": Total number of http sessions being tracked by the server at the end of the interval. This includes both active and inactive sessions.
36	24	SM120WIO	4	binary	"minLiveSessions": Minimum number of live http sessions during the interval.
40	28	SM120WIP	4	binary	"maxLiveSessions": Maximum number of live http sessions during the interval.

WebApplication interval section

Table 62. WebApplication interval section. There are multiple (0-n) sections per record. The WebApplication interval section contains information about all WebApplications involved in this activity.

Offset	Offset	Name	Length	Format	Description
0	0	SM120WIQ	256	Unicode	The WebApplication name.
256	100	SM120WIR	4	binary	"numLoadedServlets": Number of servlets that were loaded. **Note:** This value might differ from the number of servlet sections in this record since servlets might exist that have been inactive during the interval.
260	104	SM120WIS	4	binary	Number of servlet triplets in this web application section.
The following triplet appears 0-n times, once for each servlet section.					
264	108	SM120WIT	4	binary	Offset to servlet section from the beginning of this WebApplication section.
268	10C	SM120WIU	4	binary	Length of the servlet section.
272	110	SM120WIV	4	binary	Number of servlet section.

Servlet section

Table 63. Servlet section. There are multiple (0-n) sections per WebApplication section. The Servlet activity section contains information about all servlets involved per WebApplication in this activity.

Offset	Offset	Name	Length	Format	Description
0	0	SM120WIW	256	Unicode	The servlet name.
256	100	SM120WIX	4	binary	"totalRequests": Number of times the servlet service was requested during the interval.
260	104	SM120WIY	4	binary	"responseTime": Average response time in milliseconds.
264	108	SM120WIZ	4	binary	"minResponseTime": Minimum response time in milliseconds.
268	10C	SM120WJ1	4	binary	"maxResponseTime": Maximum response time in milliseconds.
272	110	SM120WJ2	4	binary	"numErrors": The number of errors that were encountered during servlet execution.
276	114	SM120WJ3	16	EBCDIC	"loadedSince": Timestamp when the servlet was loaded. Sample: Fri May 25 08:42:25 EDT 2001
292	124	SM120WJ4	8	binary	Average cpu time in microseconds.
300	12C	SM120WJ5	8	binary	Minimum cpu time in microseconds.
308	134	SM120WJ6	8	binary	Maximum cpu time in microseconds.

SMF Subtype 9: Request Activity record:

The purpose of the Request Activity SMF record is to record activity that is running inside the product. This record is produced whenever a server receives a request.

When you do capacity planning, you need to look at the costs that are involved in running requests and how many requests you process over a set period of time. You can use the SMF Subtype 9 record to monitor which requests are associated with which applications, how many requests you get, and how much resource each request uses. You can also use this record to identify the application involved, and

the CPU time that the request consumes. Because a new record is created for each request, you can determine the number of requests that you get over a specific length of time.

After you collect these SMF records for awhile, you should be able to project your future system requirements. For example, you might look at the data that was collected for a specific application, and project what your CPU requirements will be as the number of users accessing that application increases. The data that was collected might also be useful if you are charging a third party to use this application, because the record indicates the resources that were used and who used them.

The default Subtype 9 record contains all of the information that you should need to properly monitor the performance of your Enterprise JavaBeans(EJB), and web applications. You can specifically request other data, such as the formatted time stamp data, security data, or CPU usage data. However, collecting that data adds to the system overhead that is required to collect the data that populates these sections of the record.

You can activate this record through the administrative console by setting **server_SMF_request_activity_enabled=1 (or server_SMF_request_activity_enabled=true)**.

If you do not want these records to be generated, you can set **server_SMF_request_activity_enabled=0 (or server_SMF_request_activity_enabled=false)**, which turns off the creation of this SMF record type. This is the default value for this property.

Request activity record schema

The record header is the same for every Subtype 9 record that is created by the same controller. The following triplets section appears for every record that the controller generates.

The Request Activity record is divided into the following sections.

Platform neutral server information section

Table 64. Platform neutral server information section. This section contains information about the server that handled the request.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM1209BG	4	binary	The version of the Server information
4	4	SM1209BH	8	EBCDIC	Cell short name
12	C	SM1209BI	8	EBCDIC	Node short name
20	14	SM1209BJ	8	EBCDIC	Cluster short name
28	1C	SM1209BK	8	EBCDIC	Server short name
36	24	SM1209BL	4	EBCDIC	Server or controller PID
40	28	SM1209BM	1	binary	Product version level (the w in the format w.x.y.z)
41	29	SM1209BN	1	binary	Product release level (the x in the format w.x.y.z)
42	2A	SM1209BO	1	binary	Part of the product modification level (the y in the format w.x.y.z)
43	2B	SM1209BP	1	binary	Part of the product modification level (the z in the format w.x.y.z)
44	2C	*	32		Reserved

z/OSserver information section

Table 65. z/OSserver information section. This section contains information about the controller and servant where the request was dispatched. One of these sections in included in each record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM1209BQ	4	binary	The version of the server information
4	4	SM1209BR	8	EBCDIC	The name of the system on which the product is running (CVTSNAME)
12	C	SM1209BS	8	EBCDIC	The name of the sysplex on which the product is running
20	14	SM1209BT	8	EBCDIC	The job name for the controller
28	1C	SM1209BU	8	EBCDIC	The job ID for the controller
36	24	SM1209BV	8	binary	The STOKEN for the controller
44	2C	SM1209BW	2	binary	Controller ASID
46	2E	SM1209GE	2	binary	Contains the following flags: • SMF SM1209GF (bit 1) CPU Usage Overflow if turned on, the CPU usage section exceeded 30, which is the maximum number of sections that are allowed. Some of your data was lost. • SMF1209GG (bit 2) CEEGMTO failed/unavailable if turned on, the GMT offsets failed to be retrieved from the CEEGMTO API or the CEEGMTO API was not available. • Bits 3-16 are reserved
48	30	SM1209BX	20	binary	The cluster UUID
68	44	SM1209BY	20	binary	The server UUID
88	58	SM1209BZ	8	EBCDIC	The daemon group name
96	60	SM1209CA	4	binary	The hours portion of the LE GMT offset. The value is obtained from the CEEGMTO API if you are running in 31-bit mode. The field contains all zeros if the CEEGMTO API fails or is unavailable, or if you are running in 64-bit mode. The CEEGMTO API is not supported in 64-bit mode. In these situations, flag SM1209FJ is turned on to indicate that the zeros in this field are not valid GMT offsets.
100	64	SM1209CB	4	binary	The minutes portion of the LE GMT offset. The value is obtained from the CEEGMTO API if you are running in 31-bit mode. The field contains all zeros if the CEEGMTO API fails or is unavailable, or if you are running in 64-bit mode. The CEEGMTO API is not supported in 64-bit mode. In these situations, flag SM1209FJ is turned on to indicate that the zeros in this field are not valid GMT offsets.
104	68	SM1209CC	8	binary	The seconds portion of the LE GMT offset. The value is obtained from the CEEGMTO API if you are running in 31-bit mode. The field contains all zeros if the CEEGMTO API fails or is unavailable, or if you are running in 64-bit mode. The CEEGMTO API is not supported in 64-bit mode. In these situations, flag SM1209FJ is turned on to indicate that the zeros in this field are not valid GMT offsets.
112	70	SM1209CD	8	binary	The system GMT offset. The value is obtained from the CVTLDTO API.
120	78	SM1209HV	16	EBCDIC	The service level (expanded)
120	78	SM1209CE	8	EBCDIC	The service level
136	88	*	20	Reserved	

Platform neutral request information section

Table 66. Platform neutral request information section. This section provides request information that is not platform specific.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM1209CF	4	binary	The version of the request information
4	4	SM1209CG	4	binary	The PID of the dispatch servant
8	8	SM1209CH	8	binary	The ID of the dispatched task. This value is returned from pthread_self.
16	10	SM1209CI	8	binary	The amount of CPU time, in microseconds, that is used by dispatch TCB. This field might contain a negative value if field SM1209CJ contains a value other than 0.
24	18	SM1209CJ	4	binary	The completion minor code. A value of 0indicates that the request successfully completed. If a value other than 0is present, a problem occurred during processing of the request.
28	1C	*	4		Reserved
32	20	SM1209CK	4	binary	The type of request that was processed: 0indicates that the request type is not known 1indicates that the request was an IIOP request 2indicates that the request was an HTTP request 3indicates that the request was an HTTPS request 4indicates that the request was a MDB Plan "A" request. A Plan "A" request is an MDB request from a listener port that is listening in the controller. 5indicates that the request was a MDB Plan "B" request. A Plan "B" request is an MDB request from a listener port that is listening in the servant. 6indicates that the request was a MDB Plan "C" request. A Plan "C" request is an MDB request from an activation specification that is listening in the adjunct. 7indicates that the request was a SIP request 8indicates that the request was a SIPS request 9indicates that the request was an MBean request 10indicates that the request was an OTS request 11indicates that the request was an internal request 12indicates that the request was an Optimized Local Adapters (OLA) request.
36	24	*	32		Reserved

z/OSrequest information section

gotcha: There are several field descriptions within the following table that reference the z/OS WLM IWMEQTME API. You should refer to your z/OS documentation for more specific information about the content of these fields.

Table 67. z/OSrequest information section. zIIP and zAAP enclaves are not supported on z/OSVersion 1.7. Therefore, if you are running the product on z/OSVersion 1.7, fields that normally contain zIIP and zAAP enclave information, contain a value of -1.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM1209CL	4	binary	The version of the request information
4	4	SM1209CM	16	S390STCKE	The time that the request was received
20	14	SM1209CN	16	S390STCKE	The time that the request was added to the queue
36	24	SM1209CO	16	S390STCKE	The time that the request was dispatched
52	34	SM1209CP	16	S390STCKE	The time that the dispatch completed
68	44	SM1209CQ	16	S390STCKE	The time that the controller finished processing the request response
84	54	SM1209CR	8	EBCDIC	The job name for the dispatch servant
92	5C	SM1209CS	8	EBCDIC	The job ID for the dispatch servant
100	64	SM1209CT	8	binary	The STOKEN for the dispatch servant
108	6C	SM1209CU	2	binary	The ASID for the dispatch servant
110	6E	*	2		Reserved for alignment
112	70	SM1209CV	4	binary	The address of the dispatch TCB
116	74	SM1209CW	16	binary	The TTOKEN for the dispatch TCB
132	84	SM1209CX	8	binary	The amount of CPU time that was spent on non-standard CPs, such as the System z Application Assist Processor (zAAP) and z9 Integrated Information Processor (zIIP). This value is obtained from the TIMEUSED API. A value of -1 displays in this field if • A value cannot be obtained from the TIMEUSED service. • The level of z/OS on which you are running is not Version 1.9 with APAR OA20758 applied, or Version 1.10 or higher. This field might also contain a negative value if field SM1209CJ contains a value other than `0`.
140	8C	SM1209CY	8	binary	The enclave token
148	94	SM1209CZ	32		Reserved
180	B4	SM1209DA	8	binary	The enclave CPU time at the end of the dispatch of this request, as reported by the CPUTIME parameter of the IWMEQTME API. The units are in TOD format.
188	BC	SM1209DB	8	binary	The enclave zAAP CPU time at the end of the dispatch of this request, as reported by the ZAAPTIME parameter of the IWMEQTME API. The value is zero if the PTF for z/OS APAR OA22160 is not installed on your system.
196	C4	SM1209DC	8	binary	The amount of CPU time at the end of the dispatch of this request that is spent on a regular CP that could have been run on a zAAP, but the zAAP was not available. This value is obtained from the ZAAPONCPTIME. field in the IWMEQTME macro. The value is zero if the PTF for z/OS APAR OA22160 is not installed on your system.
204	CC	SM1209DD	8	binary	The zIIP enclave that is on the CPU at the end of the dispatch of this request. This value is obtained from the ZIIPONCPTIME. field in the IWMEQTME macro. The value is zero if the PTF for z/OS APAR OA22160 is not installed on your system.
212	D4	SM1209DE	8	binary	The zIIP Quality Time enclave that was on the CPU at the end of the dispatch of this request. This value is obtained from the ZIIPQUALTIME field in the IWMEQTME macro. The value is zero if the PTF for z/OS APAR OA22160 is not installed on your system.

Table 67. z/OSrequest information section (continued). zIIP and zAAP enclaves are not supported on z/OSVersion 1.7. Therefore, if you are running the product on z/OSVersion 1.7, fields that normally contain zIIP and zAAP enclave information, contain a value of -1.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
220	DC	SM1209DF	8	binary	The eligible zIIP enclave that is on the CPU at the end of the dispatch of this request. This value is obtained from the ZIIPTIME field in the IWMEQTME macro. The value is zero if the PTF for z/OS APAR OA22160 is not installed on your system.
228	E4	SM1209DG	4	binary	The zAAP normalization factor at the end of the dispatch of this request. This value is obtained from the ZAAPNFACTOR parameter of the IWMEQTME API. The value is zero if the PTF for z/OS APAR OA22160 is not installed on your system.
232	E8	SM1209DH	8	binary	The amount of CPU time that was used by the enclave as reported by the CPUTIME parameter of the IWM4EDEL API
240	F0	SM1209DI	8	binary	The delete zAAP CPU enclave. A value of 0indicates that the enclave was not deleted or not normalized. This value is obtained from the ZAAPTIME field in the IWM4EDEL macro.
248	F8	SM1209DJ	4	binary	The enclave delete zAAP normalization factor as reported by the ZAAPNFACTOR parameter of the IWM4EDEL API.
252	FC	*	4		Reserved
256	100	SM1209DK	8	EBCDIC	The enclave delete zIIP time accumulated by the enclave as reported by the ZIIPTIME parameter of the IWM4EDEL API. A value of 0 indicates that the enclave was not deleted.
264	108	SM1209DL	8	EBCDIC	The enclave delete zIIP Service accumulated by the enclave as reported by the ZIIPSERVICE parameter of the IWM4EDEL API. A value of 0 indicates that the enclave was not deleted or not normalized.
272	110	SM1209DM	8	EBCDIC	The enclave delete zAAP Service accumulated by the enclave as reported by the ZAAPSERVICE parameter of the IWM4EDEL API. A value of 0 indicates that the enclave was not deleted.
280	118	SM1209DN	8	EBCDIC	The enclave delete CPU service accumulated by the enclave as reported by the CPUSERVICE parameter of the IWM4EDEL API. A value of 0 indicates that the enclave was not deleted.
288	120	SM1209DO	4	EBCDIC	The enclave delete Response Time ratio as reported by the RESPTIME_RATIO parameter of the IWM4EDEL API. A value of 0 indicates that the enclave was not deleted.
292	124	SM1209DP	12		Reserved for alignment
304	130	SM1209DQ	73	binary	The global transaction ID (GTID) value
377	179	*	3		Reserved for alignment
380	17C	SM1209DR	4	binary	The dispatch timeout value
384	180	SM1209DS	8	EBCDIC	The transaction class, if one is being used

Table 67. z/OSrequest information section (continued). zIIP and zAAP enclaves are not supported on z/OSVersion 1.7. Therefore, if you are running the product on z/OSVersion 1.7, fields that normally contain zIIP and zAAP enclave information, contain a value of -1.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
392	188	SM1209DT	4	binary	Is either blank or contains the following flags: SM1209DU (bit 1) - if turned on, an enclave was created by this server for this request SM1209DV (bit 2) - if turned on, the timeout value was given to the product by an external source instead of being taken from the configuration for the server SM1209DW (bit 3) - if turned on, the transaction class value was given to the product by an external source instead of being taken from the configuration for the server SM1209DX (bit 4) - if turned on, this is a one way IIOP request, for which a response is not expected SM1209FJ (bit 7) - CEEGMTO failed/unavailable ((Use of SM1209GG is preferred because this section may not always be present.) SM1209FK (bit 8) - if turned on, the classification_only_trace Reliability Availability and Serviceability (RAS) attribute indicated that classification level tracing is in effect for the application server. If you issued a modify TRACERECORD,OFF command, then the classification_only_trace is not in effect. The field is valid only if the SM1209CL field is greater than or equal to 2. SM1209FM (bit 9) - if turned on, the server wide environment variable or the SMF_request_activity_enabled RAS attribute indicated to collect an SMF 120 subtype 9 record. If you issued a modify command to stop the collection, the SMF 120 subtype 9 record was not collected. The field is valid only if the SM1209CL field is greater than or equal to 2. SM1209FN (bit 10) - if turned on, the server wide environment variable or the SMF_request_activity_timestamps RAS attribute indicated to include the time stamp section in the SMF 120 subtype 9 record. If you issued a modify command to turn off the timestamp section, the SMF record does not contain the timestamp section. The field is valid only if the SM1209CL field is greater than or equal to 2.
392	188	SM1209DT	4	binary	(continued) SM1209FO (bit 11) - if turned on, the server wide environment variable or the SMF_request_activity_security RAS attribute indicated to include the security data section in the SMF 120 subtype 9 record. If you issued a modify command to turn off the security data section, the SMF record does not contain the security data section. The field is valid only if the SM1209CL field is greater than or equal to 2. SM1209FP (bit 12) - if turned on, the server wide environment variable or the SMF_request_activity_CPU_detail RAS attribute indicated to include the CPU usage breakdown section in the SMF 120 subtype 9 record. If you issued a modify command to turn off the CPU usage breakdown section, the SMF record does not contain the CPU usage breakdown section. The field is valid only if the SM1209CL field is greater than or equal to 2. SM1209FQ (bit 13) - if turned on, the propagate_transaction_name attribute indicated to use the Customer Information Control System (CICS) transaction name as the workload management (WLM) transaction class for the optimized local adapter request. The field is valid only if the SM1209CL field is greater than or equal to 2. Bits 14 - 32 are reserved
396	18C	*	32		Reserved

Table 67. z/OSrequest information section (continued). zIIP and zAAP enclaves are not supported on z/OSVersion 1.7. Therefore, if you are running the product on z/OSVersion 1.7, fields that normally contain zIIP and zAAP enclave information, contain a value of -1.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
428	1AC	SM1209FR	4	binary	The numerical value corresponds to the value of the stalled_thread_dump_action RAS attribute: 0for none, 1for javacore, 2for heapdump, 3for traceback, 4for svcdump, and 5for javatdump. The field is valid only if the SM1209CL field is greater than or equal to 2.
432	1B0	SM1209FS	4	binary	The numerical value corresponds to the value of the cputimeused_dump_action RAS attribute: 0for none, 1for javacore, 2for heapdump, 3for traceback, 4for svcdump, and 5for javatdump. The field is valid only if the SM1209CL field is greater than or equal to 2.
436	1B4	SM1209FT	4	binary	The numerical value corresponds to the value of the dpm_dump_action RAS attribute: 0for none, 1for javacore, 2for heapdump, 3for traceback, 4for svcdump, and 5for javatdump. The field is valid only if the SM1209CL field is greater than or equal to 2.
440	1B8	SM1209FU	4	binary	The numerical value corresponds to the value of the timeout_recovery RAS attribute: 1for servantand 2for session. The field is valid only if the SM1209CL field is greater than or equal to 2.
444	1BC	SM1209FV	4	binary	The value of the dispatch_timeout classification RAS attribute. The field is valid only if the SM1209CL field is greater than or equal to 2.
448	1C0	SM1209FW	4	binary	Queue timeout, which is calculated using the dispatch_timeout and queue_timeout_percent classification RAS attributes. The field is valid only if the SM1209CL field is greater than or equal to 2.
452	1C4	SM1209FX	4	binary	The value of the request_timeout classification RAS attribute. The field is valid only if the SM1209CL field is greater than or equal to 2.
456	1C8	SM1209FY	4	binary	The value of the cputimeused_limit classification RAS attribute. The field is valid only if the SM1209CL field is greater than or equal to 2.
460	1CC	SM1209FZ	4	binary	The value of the dpm_interval classification RAS attribute. The field is valid only if the SM1209CL field is greater than or equal to 2.
464	1D0	SM1209GA	8	EBCDIC	The value of the message_tag classification RAS attribute. The field is valid only if the SM1209CL field is greater than or equal to 2.
472	1D8	SM1209GH	4	binary	Length of obtained affinity RNAME
476	1DC	SM1209GI	128	EBCDIC	Obtained affinity RNAME
604	25C	SM1209GJ	4	binary	Length of routing affinity RNAME
608	260	SM1209GK	128	EBCDIC	Routing affinity RNAME

z/OSformatted timestamps section

This section contains the date and time information for specific events that occured durring the processing of the request. All of the times that are included in this section are expressed in the format yyyy/mm/dd hh:mm:ss.xxxxxx, where yyyy is the year, mm is the month, dd is the day, hh is the hour, mm is the minutes, ss is the seconds, and xxxxxx is the microseconds.

Including the timestamp section in the Subtype 9 record is optional. Collecting the data to update this section adds system overhead and can make these SMF records larger. Therefore, the collection of this data, by default, is turned off. When the collection of this data is turned off, the Number of records field in the triplets section, that is located at the beginning of the record contains, a zero.

Table 68. z/OSformatted timestamps section. To turn on the collection of this data, use the administrative console to specify either the **server_SMF_request_activity_timestamps=1***or* **server_SMF_request_activity_timestamps=true***SMF property.*

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM1209EA	26	EBCDIC	The time that the request was received
26	1A	SM1209EB	26	EBCDIC	The time that the request was added to the WLM queue
52	34	SM1209EC	26	EBCDIC	The time that the request was dispatched in the servant
78	4E	SM1209ED	26	EBCDIC	The time that the dispatch completed in the servant
104	68	SM1209EE	26	EBCDIC	The time that the controller finished processing the request
130	82	*	2		Reserved for alignment

Network data for HTTP, SIP, and IIOP transports section

Table 69. Network data for HTTP, SIP, and IIOP transports section. This section contains information about the origin of the request that this record describes. It is only present for protocols for which the product can obtain origin information. For example, this section does not exist for Message Driven Beans (MDBs) requests. A record contains only one instance of this section.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM1209EF	4	binary	The version of the network data
4	4	SM1209EG	8	binary	The size of the request, in bytes, that was received from the client
12	C	SM1209EH	8	binary	The size of the response, in bytes, that is sent back to the client
20	14	SM1209EI	4	binary	The target port for the request. A value of -1indicates that local communications was used.
24	18	SM1209EJ	4	binary	The length of the origin string
28	1C	SM1209EK	128	EBCDIC	The origin string. Following is an example of an origin string: `ip addr=9.57.7.193 port=1344`.The bytes that follow the string contain blank spaces.
156	9C	*	32		Reserved

Classification data section

Table 70. Classification data section. This section contains the classification information for this request. If a transaction class was encountered earlier, this information might have been used to determine that transaction class name.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM1209EL	4	binary	The version of the classification data
4	4	SM1209EM	4	binary	The data type. Not all of these data types apply to all requests. For example, only data types 6, 7, or 8 appear in this field for an HTTP request. 1indicates that it is the name of an application 2indicates that it is the name of a module 3indicates that it is the name of a component 4indicates that it is the name of a class 5indicates that it is the name of a method 6indicates that it is a URI 7indicates that it is the name of the target host 8indicates that it is the name of the target port 9indicates that it is a message listener port 10indicates that it is the name of a selector 11indicates that it is the name of the Optimized Local Adapters (OLA) service 12indicates that it is the CICS imported transaction name
8	8	SM1209EN	4	binary	The length of the data
12	C	SM1209EO	128	EBCDIC	The data string

Security data section

This section contains the security information for each request. There is a separate security data section for each identity type. Depending on your security configuration, up to three identity types might exist. Therefore, there can be up to three instances of this section in a record, depending on which data is available for the request, for which the report is generated.

Including the security sections in the Subtype 9 record is optional. Collecting the data to update this section adds system overhead and can make these SMF records larger. Therefore, the collection of this data, by default, is turned off. When the collection of this data is turned off, the Number of records field in the triplets section at the top of the record contains a zero.

Table 71. Security data section. To turn on the collection of this data, use the administrative console to specify either the **server_SMF_request_activity_security=1** *or* **server_SMF_request_activity_security=true** *SMF property.*

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM1209EP	4	binary	The version of the security data

Table 71. Security data section (continued). To turn on the collection of this data, use the administrative console to specify either the **server_SMF_request_activity_security=1***or* **server_SMF_request_activity_security=true***SMF property.*

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
4	4	SM1209EQ	4	binary	The data type: 1indicates that it is the server identity 2indicates that it is the received identity 3indicates that it is the invocation identity
8	8	SM1209ER	4	binary	The length of the identity
12	C	SM1209ES	64	EBCDIC	The identity string

CPU usage breakdown section

This section contains information about an item that was called and the CPU time that the task consumed, minus the time that the CPU spent waiting for tasks it initiated to complete. This calculation is different from the way CPU time is calculated in the container records.

There can be up to 30 instances of this section in a record; one for each item that is called. If your application calls more than 30 different items under the dispatch of a single request, only the first 30 items are included. Bit 5 of field SM1209DT indicates when such a truncation occurs.

Including the CPU usage section in the Subtype 9 record is optional. Collecting the data to update this section adds system overhead and can make these SMF records quite large. Therefore, the collection of this data, by default, is turned off. When the collection of this data is turned off or none were collected during a request, the field for the number of CPU usage break down sections (SM1209AB) in the triplet at the top of the record contains a zero.

Table 72. CPU usage breakdown section. To turn on the collection of this data, use the administrative console to specify either the **server_SMF_request_activity_CPU_detail=1***or* **server_SMF_request_activity_CPU_detail=true***SMF property.*

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM1209ET	4	binary	The version of the CPU usage data
4	4	SM1209EU	4	binary	The data type: 1indicates that the data comes from the EJB container 2indicates that the data comes from the web container
8	8	SM1209EV	8	binary	The amount of CPU time, in microseconds, that the item, such as an EJB or a servlet, spent in dispatch
16	10	SM1209FI	8	binary	The elapsed time, in milliseconds, that is spent processing the item, such as an EJB or servlet
24	18	SM1209EW	4	binary	How many times the item, such as an EJB or a servlet, was executed during the dispatch of this request
28	1C	SM1209EX	4	binary	The length of string 1
32	20	SM1209EY	256	EBCDIC	String 1. String 1 has one of the following values: AMC, which indicates that an EJB was processed Web App, which indicates that a Servlet was processed
288	120	SM1209EZ	4	binary	The length of string 2

Table 72. CPU usage breakdown section (continued). To turn on the collection of this data, use the administrative console to specify either the **server_SMF_request_activity_CPU_detail=1***or* **server_SMF_request_activity_CPU_detail=true***SMF property.*

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
292	124	SM1209FA	256	EBCDIC	String 2 has one of the following values: The method name or signature, if an EJB is accessing the data The name of the servlet if a servlet is accessing the data

User data section

You can use the package com.ibm.websphere.smf API to add up to 5 User data sections to the end of this record. Each of these sections must be less than or equal to 2 KB in length. The data that is contained in these sections is not formatted and appears exactly as it is received from your application.

Table 73. User data section. The SMF 120 Subtype 9 record can be turned on and off dynamically. Use the SmfEventNotifier API, if you want to be notified when the product starts and stops writing this record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM1209FE	4	binary	The version of the User data section
4	4	SM1209FF	4	binary	The user data type. Types 65535 and lower are reserved for IBMuse.
8	8	SM1209FG	4	binary	The length of the User data section
12	C	SM1209FH	2048	binary	The data that the application added

Asynchronous data section

This section contains information about requests that the server runs asynchronously.

An asynchronous request has one instance of this section. This section is not applicable for non-asynchronous requests.

Table 74. Asynchronous data section. The SMF 120 subtype 9 record can be turned on and off dynamically. To turn on the collection of this data, use the administrative console to specify either the server_SMF_request_activity_async=1 or the server_SMF_request_activity_async=true SMF property.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM1209GM	4	binary	The version of the asynchronous data section.
4	4	SM1209GL	16	S390STCKE	The time that the execution context was created.
20	14	SM1209GN	16	S390STCKE	The time that the execution started.
36	24	SM1209GO	16	S390STCKE	The time that the execution completed.
52	34	SM1209GP	4	binary	The servant process ID.
56	38	SM1209GQ	8	EBCDIC	The servant job name.
64	40	SM1209GR	8	EBCDIC	The servant job ID.
72	48	SM1209GS	8	binary	The servant token.
80	50	SM1209GT	2	binary	The servant ASID.

Table 74. Asynchronous data section (continued). The SMF 120 subtype 9 record can be turned on and off dynamically. To turn on the collection of this data, use the administrative console to specify either the server_SMF_request_activity_async=1 or the server_SMF_request_activity_async=true SMF property.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
82	52	*	2		The reserved alignment.
84	54	SM1209GU	8	binary	The execution context task ID.
92	5C	SM1209GV	4	binary	The execution context TCB address.
96	60	SM1209GW	16	binary	The execution context TCB TToken.
112	70	SM1209GX	8	binary	The dispatch task ID.
120	78	SM1209GY	4	binary	The dispatch TCB address.
124	7C	SM1209GZ	16	binary	The dispatch TCB TToken.
140	8C	SM1209HA	8	binary	The execution context enclave token.
148	94	SM1209HB	8	binary	The dispatch enclave token.
156	9C	SM1209HC	8	EBCDIC	The transaction class used to create the enclave.
164	A4	SM1209HD	4	binary	Contains the following flags: • SM1209HE in bit 1. A value of 0indicates that the enclave was joined. A value of 1indicates that the enclave was created. • SM1209HF in bit 2. A value of 1indicates that the enclave was scheduled with the daemon. • Reserved, for bits 3 through 32.
168	A8	SM1209HG	8	binary	The enclave CPU so far.
176	B0	SM1209HH	8	binary	The enclave zAAP CPU so far.
184	B8	SM1209HI	8	binary	The enclave zAAP eligible on the CP.
192	C0	SM1209HJ	8	binary	The enclave zIIP on the CPU so far.
200	C8	SM1209HK	8	binary	The enclave zIIP quality time so far.
208	D0	SM1209HL	8	binary	The Enclave zIIP CPU so far.
216	D8	SM1209HM	4	binary	The zAAP normalization factor.
220	DC	*	4		The reserved alignment.
224	E0	SM1209HN	8	binary	The dispatch CPU.
232	E8	SM1209HO	8	binary	The dispatch CPU offload to non-standard CPs, for example zAAP or zIIP.
240	EC	SM1209HP	4	binary	Length of the work class name.
244	F4	SM1209HQ	128	EBCDIC	The work package or class name.
372	174	SM1209HR	4	binary	The length of the work manager name.
376	178	SM1209HS	128	EBCDIC	The work manager name.
504	1F8	SM1209HT	4	binary	The identity length.
508	1FC	SM1209HU	64	EBCDIC	The identity.
572	23C	*	16		Reserved.

SMF Subtype 10: Outbound Request record:

The purpose of the Outbound Request SMF record is to record requests that go outbound. This record is produced whenever an optimized local adapter request goes outbound.

You can use the SMF Subtype 10 record to monitor how many outbound requests are occurring, where they are going and how long it takes.

You can activate this record through the administrative console by setting **server_SMF_outbound_enabled=1 (or server_SMF_outbound_enabled=true)**.

If you do not want these records to be generated, you can set **server_SMF_outbound_enabled=0 (or server_SMF_outbound_enabled=false)**, which turns off the creation of this SMF record type. This is the default value for this property.

Outbound Request record schema

The record header is the same for every Subtype 10 record that is created by the same server.

The Outbound Request record is divided into the following sections.

Platform neutral server information section

Table 75. Platform neutral server information section. This section contains information about the server that handled the request.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120ABG	4	binary	The version of the server information
4	4	SM120ABH	8	EBCDIC	Cell short name
12	C	SM120ABI	8	EBCDIC	Node short name
20	14	SM120ABJ	8	EBCDIC	Cluster short name
28	1C	SM120AAK	8	EBCDIC	Server short name
36	24	SM120ABL	4	EBCDIC	Server or controller PID
40	28	SM120ABM	1	binary	Product version level (the w in the format w.x.y.z)
41	29	SM120ABN	1	binary	Product release level (the x in the format w.x.y.z)
42	2A	SM120ABO	1	binary	Part of the product modification level (the y in the format w.x.y.z)
43	2B	SM120ABP	1	binary	Part of the product modification level (the z in the format w.x.y.z)
44	2C	*	32		Reserved

z/OSserver information section

Table 76. z/OSserver information section. This section contains information about the controller and servant where the request was dispatched. One of these sections in included in each record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120ABQ	4	binary	The version of the server information
4	4	SM120ABR	8	EBCDIC	The name of the system on which the product is running (CVTSNAME)
12	C	SM120ABS	8	EBCDIC	The name of the sysplex on which the product is running
20	14	SM120ABT	8	EBCDIC	The job name for the controller

Table 76. z/OSserver information section (continued). This section contains information about the controller and servant where the request was dispatched. One of these sections in included in each record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
28	1C	SM120ABU	8	EBCDIC	The job ID for the controller
36	24	SM120ABV	8	binary	The STOKEN for the controller
44	2C	SM120ABW	2	binary	Controller ASID
46	2E	SM120AGE	2	binary	Contains the following flags: • .Bit 1 is reserved • SMF120AGG (bit 2) CEEGMTO failed/unavailable if turned on, the GMT offsets failed to be retrieved from the CEEGMTO API or the CEEGMTO API was not available. • Bits 3-16 are reserved
48	30	SM120ABX	20	binary	The cluster UUID
68	44	SM120ABY	20	binary	The server UUID
88	58	SM120ABZ	8	EBCDIC	The daemon group name
96	60	SM120ACA	4	binary	The hours portion of the LE GMT offset. The value is obtained from the CEEGMTO API if you are running in 31-bit mode. The field contains all zeros if the CEEGMTO API fails or is unavailable, or if you are running in 64-bit mode. The CEEGMTO API is not supported in 64-bit mode. In these situations, flag SM120AGG is turned on to indicate that the zeros in this field are not valid GMT offsets.
100	64	SM120ACB	4	binary	The minutes portion of the LE GMT offset. The value is obtained from the CEEGMTO API if you are running in 31-bit mode. The field contains all zeros if the CEEGMTO API fails or is unavailable, or if you are running in 64-bit mode. The CEEGMTO API is not supported in 64-bit mode. In these situations, flag SM120AGG is turned on to indicate that the zeros in this field are not valid GMT offsets.
104	68	SM120ACC	8	binary	The seconds portion of the LE GMT offset. The value is obtained from the CEEGMTO API if you are running in 31-bit mode. The field contains all zeros if the CEEGMTO API fails or is unavailable, or if you are running in 64-bit mode. The CEEGMTO API is not supported in 64-bit mode. In these situations, flag SM120AGG is turned on to indicate that the zeros in this field are not valid GMT offsets.
112	70	SM120ACD	8	binary	The system GMT offset. The value is obtained from the CVTLDTO API.
120	78	SM120AHV	16	EBCDIC	The service level (expanded)
136	88	*	20		Reserved

Outbound Request information section

Table 77. Outbound Request information section. This section contains information about the outbound request. One of these sections is included in each record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120ACF	4	binary	The version of the outbound request information
4	4	SM120ACG	4	binary	The PID of the dispatch servant
8	8	SM120ACH	8	binary	The ID of the dispatched task. This value is returned from pthread_self.

Table 77. Outbound Request information section (continued). This section contains information about the outbound request. One of these sections is included in each record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
16	10	SM120ACK	4	binary	Outbound request type **0** unknown **1** WOLA **2** OTMA
20	14	*	4		Reserved
24	18	SM120ACR	8	EBCDIC	The job name for the dispatch servant
32	20	SM120ACS	8	EBCDIC	The job ID for the dispatch servant
40	28	SM120ACT	8	binary	The STOKEN for the dispatch servant
48	30	SM120ACU	2	binary	The ASID for the dispatch servant
50	32	*	2		Reserved for alignment
52	34	SM120ACV	4	binary	The address of the dispatch TCB
56	38	SM120ACW	16	binary	The TTOKEN for the dispatch TCB
72	48	SM120ACY	8	binary	The enclave token
80	50	SM120AD1	8	binary	The number of bytes sent
88	58	SM120AD2	8	binary	The number of response bytes
96	60	SM120AD3	16	S390STCKE	The time the request went outbound
112	70	SM120AD4	16	S390STCKE	The time the outbound request returned
128	80	*	32		Reserved

WOLA Outbound Request type specific section

Table 78. WOLA Outbound Request type specific section. This section contains specific information about the WOLA outbound request. There are zero to one of these sections included in each record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120AD5	4	binary	The version of the WOLA outbound request type specific section
4	4	SM120AD6	12	EBCDIC	The register name the outbound request is going to
16	10	SM120AD7	256	EBCDIC	The service name the outbound request is going to
272	110	SM120AD8	256	binary	The outbound request correlator context. This correlator context also ends up in the CICS SMF 110 records.
528	210	SM120AD9	24		Reserved

Outbound Request transaction context section

Table 79. Outbound Request transaction context section. This section contains transaction information about the WOLA outbound request. There are zero to one of these sections included in each record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120ADA	4	binary	The version of the outbound request transaction context section

Table 79. Outbound Request transaction context section (continued). This section contains transaction information about the WOLA outbound request. There are zero to one of these sections included in each record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
4	4	SM120ADB	140	binary	The transactional XID
144	90	SM120ADC	24		Reserved

Outbound Request security context section

Table 80. Outbound Request security context section. This section contains security information about the WOLA outbound request. There are zero to one of these sections included in each record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120ADD	4	binary	The version of the outbound request security context section
4	4	SM120ADE	8	EBCDIC	The security context
12	C	SM120ADF	28		Reserved

Outbound Request CICScontext section

Table 81. Outbound Request CICScontext section. This section contains the CICScontext associated with the WOLA outbound request. There are zero to one of these sections included in each record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120ADG	4	binary	The version of the outbound request CICScontext section
4	4	SM120ADH	80	binary	The CICScontext
84	54	SM120ADI	28		Reserved

OTMA Outbound Request type specific section

Table 82. OTMA Outbound Request type specific section. This section contains the OTMA context associated with the WOLA outbound request. There are zero to one of these sections included in each record.

Offset (decimal)	Offset (hexadecimal)	Name	Length	Format	Description
0	0	SM120ADJ	4	binary	The version of the OTMA outbound request type specific section
4	4	SM120ADK	12	EBCDIC	The OTMA register name of the outbound request
16	10	SM120ADL	256	EBCDIC	The OTMA service name of the outbound request
272	110	SM120ADM	8	EBCDIC	The OTMA IMS™transaction name of the outbound request
280	118	SM120ADN	8	EBCDIC	The OTMA IMSgroup id of the outbound request
288	120	SM120ADO	16	EBCDIC	The OTMA IMSserver name of the outbound request
304	130	SM120ADP	24		Reserved

Chapter 16. Choosing diagnostic information sources

You can use a variety of diagnostic information sources to view application data and troubleshoot problems.

About this task

Troubleshooting problems in a complex server environment can be challenging, considering the many choices of diagnostic information to analyze and wide variety of potential problem areas. Familiarize yourself with the different types of diagnostic tools and information that the product provides to maximize your efficiency and productivity when you are confronted with a problem.

Procedure

Read the following topics for information about specific sources of diagnostic data, and the tools or resources you might need to view or work with that data.

Table 83. Type of diagnostic tools and data. Click the links to read about more information.

Type of diagnostic tools or data:	Notes® and instructions for use appear in:
CEEDUMPs	"CEEDUMPs in the job log"
SVC dumps	Viewing SVC dumps
CTRACE and JRas data	Viewing CTRACE and JRas data through IPCS
Error log data	Viewing error log contents through the Log Browse Utility (BBORBLOG)
z/OS display command	Using the z/OS display command
Java minor codes	Converting Java minor codes
SYSPRINT	Diagnosing problems with message logs
Message routing	Message routing

CEEDUMPs in the job log

An error caught by LE or the Java runtime can result in the production of a CEEDUMP, which is written to a separate CEEDUMP specification in the job log.

To view the dump contents, select the CEEDUMP portion of the output for the address space. The 'Traceback' section at the beginning of the dump can be very helpful.

SVC dumps

A SVC dump is a core dump initiated by the operating system generally when a programming exception occurs. SVC dump processing stores data in dump data sets that you pre-allocate, or that the system allocates automatically as needed.

Alternatively, you can initiate an SVC dump through the MVS console, to gather diagnostic data for a 'hang' condition, for example. SVC dumps that you initiate this way are called console dumps.

One example of an abend that could occur is the EC3 abend. WebSphere Application Server for z/OS requests an SVC dump when a controller terminates a servant (region) with an EC3 abend when timeout conditions occur.

- Your installation can set parmlib options that determine what to dump, eliminate duplicate dumps, and so on. WebSphere Application Server for z/OS provides a dump parmlib sample in

```
SBBOJCL(BBODMCCB)
```

.

- The standard SDATA expected in a SVC dump:

  ```
  SDATA=(ALLNUC,CSA,GRSQ,LPA,LSQA,PSA,RGN,SQA,SUM,SWA,TRT),end
  ```

- If you cannot find an SVC dump for a specific abend, your installation might be using Dump analysis and elimination (DAE) to suppress the dump. If this is the case, you can change DAE to let the dump be taken or set a SLIP on the specific abend for a particular job name if the timeout is consistently happening. For further information, see:
  - z/OS MVS Diagnosis: Tools and Service Aids, GA22-7589 for details about using DAE.
  - z/OS MVS System Commands, SA22-7627 for details about the SLIP command, which controls SLIP (serviceability level indication processing), a diagnostic aid that intercepts or traps certain system events and specifies what action to take. Using the SLIP command, you can set, modify, and delete SLIP traps.
- When you initiate a console dump:
  - When you want an SVC dump of a servant region, also request a dump of the servant's controller region.
  - Unless you suspect a particular servant region as the source of a problem, dump the controller region and all of its servant regions.
- If syslog contains a message indicating that the maxspace limit was reached for this dump, the SVC dump might be a partial one that might not contain the data you need to diagnose the timeout. This limit means that the data set used for SVC dump is not large enough, and you have to change the size to capture a complete dump.

  Note: If you are running WebSphere Application Server for z/OS with 64-bit support, and you specify RGN (region) in the SDATA parameter set, you will need to allocate a much larger space limit for the SVC dump.
- To view CEEDUMP contents within the SVC dump, use the IPCS verbexit LEDATA, with the CEEDUMP or NTHREADS options, to format and analyze Language Environment control blocks. For additional information, see z/OS Language Environment Debugging Guide, GA22-756 for instructions for using IPCS to format and analyze CEEDUMP contents.

See z/OS MVS Diagnosis: Tools and Service Aids, GA22-7589 for additional information about SVC dumps.

Formatting CTRACE data with an IPCS dialog

You can set up an IPCS dialog to format applications trace data gathered by WebSphere Application Server for z/OS.

Before you begin

Once activated, the WebSphere Application Server for z/OS always writes trace data into memory buffers. The number and size of these buffers is controlled using WebSphere variables. You can get this trace data from a dump, which may be taken by the system or requested by the operator through DUMP or SLIP commands.

To view messages or application trace data from Component Trace, you must use the interactive problem control system (IPCS) to format the data. The source of the trace data can be a dump data set or a trace data set. When setting up IPCS, your installation may customize IPCS for its users.

IBM recommends providing access to the IPCS dialog through the Profile Management Tool or the **zpmt** command. If your installation has not customized IPCS as recommended, you need to start the IPCS dialog. See z/OS MVS IPCS User's Guide, SA22-7596 to find out how to start the IPCS dialog.

About this task

Perform the following steps to use the IPCS dialog to format application trace data:

Procedure

1. From the IPCS Primary Option Menu panel, select option 6 (*COMMAND*).
2. On the IPCS Subcommand Entry panel:
   a. Issue the *SETDEF* subcommand to determine the default values for routing displays.
   b. Enter the *CTRACE* command, with the following required parameters: CTRACE COMP(*cell_short_name*)

      where *cell_short_name* is the value specified through the Profile Management Tool or the **zpmt** command to identify the location of server configuration files. The name must be 8 or fewer characters and all uppercase.

      Note: If you were interested in only JRAS data, you would enter the following:

      ```
      CTRACE COMP(cell_short_name
      )USEREXIT(JRAS)
      ```

      Specify additional parameters as necessary.

      Example: To direct trace data to the terminal only, you would append the NOPRINT and TERMINAL parameters to the CTRACE command.

   Tip: For a complete list of CTRACE command parameters, see z/OS MVS IPCS Commands, SA22-7594.
3. View your application's data, basing the method you choose on which one is appropriate for the location of the data:
   - If you directed output to the IPCS print data set (IPCSPRNT), use the ISPF/PDF Browse option.
   - If you directed output to the terminal, use the Dump Display Reporter panel

   Tips: To navigate through the trace data on the Dump Display Reporter panel, use the commands and PF keys listed in z/OS MVS IPCS User's Guide, SA22-7596.

   CTRACE enables you to view multiple traces together with the trace data from the various sources intermixed based on the time stamp. See z/OS MVS IPCS Commands, SA22-7594, for specifics on using this MERGE subcommand.

Formatting CTRACE data in batch mode with IPCS

You can use the interactive problem control system (IPCS) in batch mode to automate formatting CTRACE data.

Before you begin

You must create an IPCS dump directory before you can use IPCS in batch mode. When setting up IPCS, your installation may customize IPCS for its users. This customization can include modifying the IBM-supplied BLSCDDIR CLIST with default values for creating an IPCS dump directory.

About this task

To view messages or application trace data from Component Trace, you must use the interactive problem control system (IPCS) to format the data. Using IPCS in batch mode is the easiest method of formatting data, especially if you do not have much experience with using IPCS, TSO/E and ISPF. Through batch mode, you can use IPCS to format trace data and write it to an MVS data set. Optionally, you may copy the contents of that data set into an HFS file for viewing.

When your installation has modified the BLSCDDIR CLIST the steps outlined herein will create an IPCS dump directory.

1. Decide on a fully-qualified data set name for the directory.
2. From the TSO/E command prompt, enter the BLSCDDIR command, specifying the data set name.

   For example, to create a dump directory named IBMUSER.DDIR, enter:

```
%blscddir dsn('ibmuser.ddir')
```

If your installation has not customized IPCS, you might need to alter other BLSCDDIR CLIST parameters. See the z/OS MVS IPCS User's Guide, SA22-7596 and z/OS MVS IPCS Commands, SA22-7594 for more details about using the BLSCDDIR CLIST to create a dump directory.

Perform the following steps to use IPCS in batch mode to format application trace data:

Procedure

1. Create a file and copy the following sample JCL into it. This JCL invokes IPCS to extract and format JRAS trace data and write it into an MVS data set, and then uses the TSO/E OPUT command to copy the formatted data from the MVS data set into an HFS file.

```
//IBMUSERX   JOB ,
// CLASS=J,NOTIFY=&SYSUID,MSGCLASS=H
//IPCS       EXEC PGM=IKJEFT01,REGION=4096K,DYNAMNBR=50
//IPCSDDIR   DD DSN=IBMUSER.DDIR,DISP=SHR
//IPCSDOC    DD STDERR=H
//JRASTRC    DD DSN=IBMUSER.CB390.CTRACE,DISP=SHR
//IPCSPRNT   DD DSN=IBMUSER.IPCS.OUT,DISP=OLD
//SYSTSPRT   DD STDERR=*
//SYSTSIN    DD *
IPCS
DROPDUMP DDNAME(JRASTRC)
PROFILE LINESIZE(80)PAGESIZE(99999999)
SETDEF NOCONFIRM
CTRACE COMP(SYSBBOSS) DDNAME(JRASTRC) FULL PRINT +
       NOTERMINAL
DROPDUMP DDNAME(JRASTRC)
END
/*
//OPUT       EXEC PGM=IKJEFT01,REGION=4096K,DYNAMNBR=50
//SYSTSPRT   DD STDERR=*
//SYSTSIN    DD *
oput 'ibmuser.ipcs.out' '/u/ibmuser/ipcs/jrastrace.txt' TEXT
/*
```

2. Edit the sample JCL to replace *IBMUSER.DDIR* with the data set name that you used for the IPCS dump directory you created.
   a. Use the *PAGESIZE* parameter on the *PROFILE* statement only if you do not want to print the output data set.
   b. You may replace the HFS file name with the name of an existing HFS file, but you do not have to do so. The OPUT command processing will create a new HFS file, if the one specified does not exist, and grants read and write access to that file for your user ID only.

      If you do specify an existing HFS file, the OPUT command processing will write over any data that is already in that file. If you want to know more about the OPUT command, see the z/OS UNIX System Services Command Reference, SA22-7802.
   c. Change the data set name specified on the *JRASTRC DD* in the example to the name of the data set containing the CTRACE data.
   d. Change the name of the MVS data set on both the *JRASTRC DD* statement and the *OPUT* command in the SYSTSIN stream, as necessary. The formatted output of the JRAS CTRACE data is first written to the MVS data set specified by the IPCSPRNT DD statement and then (optionally)

copied to the HFS data set. You must either pre-allocate this data set, or change the sample JCL to allocate the data set. This data set should have a record format of VBA and a record length of 133.

3. Submit the JCL to start the IPCS batch job.

What to do next

Once you are done you can use a UNIX editor, such as vi, to view your trace data in the HFS file. If you want to know more about the UNIX editors, see z/OS UNIX System Services User's Guide, SA22-7801.

CTRACE enables you to view multiple traces together with the trace data from the various sources intermixed based on the time stamp. See z/OS MVS IPCS Commands, SA22-7594, for specifics on using this MERGE subcommand.

Sample JCL to display WebSphere for z/OS trace data

Use this sample JCL to display trace data for WebSphere Application Server for z/OS.

The following sample shows JCL that displays WebSphere for z/OS trace data.

Note: The JCL uses an IPCS dump directory (in VSAM data set `userid.DUMP.DIR`) that must be allocated before you run the JCL. See z/OS MVS IPCS Commands, SA22-7594 , for information about initializing a dump directory.

```
//SHOWTRC  JOB <job card info>
//JOBLIB   DD DISP=SHR,DSN=BBO.MIGLIB
//       DD DISP=SHR,DSN=SYS1.MIGLIB
//PRINTIT EXEC PGM=IKJEFT01,REGION=0M
//IPCSDDIR DD DISP=(OLD,KEEP),DSN=userid.DUMP.DIR
//IPCSPARM DD DISP=SHR,DSN=SYS1.PARMLIB
//SYSTSPRT DD SYSOUT=*
//IPCSTOC  DD SYSOUT=*
//IPCSPRNT DD SYSOUT=*
//*--------------------------
//SYSTSIN DD *
IPCS NOPARM
  CTRACE COMP(SYSBBOSS) SUB((subname)) FULL DSN('dump.data.set')
/*
```

The following example shows JCL that displays WebSphere for z/OS trace data for multiple address spaces.

```
//SHOWTRC2 JOB <job card info>


//JOBLIB   DD DISP=SHR,DSN=BBO.MIGLIB
//       DD DISP=SHR,DSN=SYS1.MIGLIB
//PRINTIT EXEC PGM=IKJEFT01,REGION=0M
//IPCSDDIR DD DISP=(OLD,KEEP),DSN=userid.DUMP.DIR
//IPCSPARM DD DISP=SHR,DSN=SYS1.PARMLIB
//SYSTSPRT DD SYSOUT=*
//IPCSTOC  DD SYSOUT=*
//IPCSPRNT DD SYSOUT=*
//*--------------------------
//SYSTSIN DD *
IPCS NOPARM
  MERGE
  CTRACE COMP(SYSBBOSS) SUB((subname)) FULL DSN('dump.data.set')
  CTRACE COMP(SYSBBOSS) SUB((subname2)) FULL DSN('dump.data.set')
  MERGEEND
/*
```

Note: You need to copy the files from `WAS_HOME/lib/ipcs/` into BBO.MIGLIB.

ICPS CTRACE command

The CTRACE command specifies the cell from which trace data will be gathered.

Read “Formatting CTRACE data with an IPCS dialog” on page 256 for information on when to use the CTRACE command with an IPCS dialog. The CTRACE command uses the following syntax:

```
CTRACE COMP (CELL_SHORT_NAME)
```

where *CELL_SHORT_NAME* is the value specified through the Profile Management Tool or the **zpmt** command to identify the location of server configuration files. The name must be 8 or fewer characters and all uppercase.

You can also use the IPCS CTRACE command to merge multiple trace entities together such as multiple WebSphere Application Server for z/OS address space traces, OMVS, and TCPIP. For example code, see “Sample JCL to display WebSphere for z/OS trace data” on page 259.

IPCS CTRACE subname query

If the trace data set is an SVC dump, the trace subname must also be specified. This subname is the aggregation of the address space's jobname with its ASID (address space identifier), in printable hexadecimal.

An easy way to determine the subname is to query CTRACE for the data using the following IPCS subcommand:

```
CTRACE QUERY DSN('dump.data.set')
```

Once you get the subname you can view the WebSphere Application Server for z/OS trace data with the following IPCS subcommand:

```
CTRACE COMP(CELL_SHORT_NAME) SUB((subname)) FULL DSN('dump.data.set')
```

where *CELL_SHORT_NAME* is the value specified through the Profile Management Tool or the **zpmt** command to identify the location of server configuration files. The name must be 8 or fewer characters and all uppercase.

Note: The *subname* parameter is optional for only the trace data set. It is required when viewing the trace data using the dump data set.

Viewing error log contents through the Log Browse Utility (BBORBLOG)

You can use the Log Browse Utility (BBORBLOG) to view error log contents for troubleshooting and tuning purposes.

About this task

You can use the Log Browse Utility (BBORBLOG) to view the error log stream. If you need to look at the WebSphere Application Server for z/OS error log stream, use ISPF option 6 to enter the command:

Procedure

1. Use ISPF option 6 to enter the proper command.

   ```
   ex 'BBO.SBBOEXEC(BBORBLOG)' 'BBO.BOSSXXXX'
   ```

   where the log-stream name is `BBO.BOSSXXXX`
2. The space allocation and the unit for the allocation are contained within the REXX code. If you keep a large amount of trace data, the allocation must be made larger.

3. The WebSphere Application Server for z/OS provides an ISPF REXX EXEC named BBORBLOG, that allows you to browse the error log stream.
4. Save the output.

   When you use the BBORBLOG browser, it creates a data set name, which depends on the TSO PROFILE PREFIX value. If you invoke the TSO environment with a TSO PROFILE PREFIX definition value, this value is used, in conjunction with the log stream name, to form the output data set. For example, if you issue the TSO PROFILE PREFIX(HARRY) command and the log stream name is BBO.BOSSXXXX, the output data set is HARRY.BBO.BOSSXXXX.

   If you wish to save your browser output, you can use the **outdsn** parameter to specify a name for the output data set before you save it. For more information on this parameter, see the usage information on the BBORBLOG log browse utility. The contents of the current view of the log stream will remain until the stream reaches its retention date. The next time you invoke the browser, however, the current view of the log stream will be deleted because if it uses the same data set name. If you use the **outdsn** parameter to specify a unique output data set name the current view of the log stream is not deleted. The previous data will exist in another record (not the current view) until its retention date.

Results

Use the following information to determine viewing of the error log:
- By default, the macro formats the error records to fit a 3270 display.
- Timestamps are in Greenwich Mean Time (GMT) unless changed by setting the WebSphere Application Server variable `ras_time_local` to 1.
- Message BBOJ0051I, which appears in the job output, can help correlate error-log entries to the proper job output.

Using the log browse utility (BBORBLOG)

Use the log browse utility (BBORLOG) to view log stream files.

About this task

The browser takes the following parameters.

Table 84. Log browse utility parameters. The data stored in the WebSphere Application Server log stream must be formatted using the BBORBLOG Rexx Exec routine. By default, BBORBLOG formats the error records to fit a 3270 display.

Parameter	Description
log stream name	The name of the log stream. See the job messages for the name of the log stream.
format option	**80** The default. The log stream record is formatted on a lrecl length of 80 characters. Additional lines are wrapped. **NOFORMAT** Turns off formatting. The error log message displays as one log message string in the browse file.
reclen	This optional keyword parameter specifies the formatted record length for the output data set. When you use this parameter, it must follow the **log stream name** and **format option** parameters in order. However, if you specify this parameter, the **format option** is an optional positional parameter. If you specify **NOFORMAT** for the **format option** parameter, the default value is 4096. If the **format option** parameter value is 80, this parameter value defaults to 80. The value range for this parameter is 80 to 32752.

Table 84. Log browse utility parameters (continued). The data stored in the WebSphere Application Server log stream must be formatted using the BBORBLOG Rexx Exec routine. By default, BBORBLOG formats the error records to fit a 3270 display.

Parameter	Description
outdsn	This optional keyword parameter is the non-fully qualified data set name for the output data set. When you use this parameter, it must follow the **log stream name** and **format option** parameters in order. The default value for this parameter is the name of the log stream.

Procedure

1. Edit BBO.SBBOEXEC(BBORBLOG) and set the *bborblogpath* variable to the HFS path where the `bborblog dll` exists or copy the `bborblog dll` from the HFS into a dataset that is in linklist. For example, copy the `cp -X bborblog "//'xxx.LOAD(bborblog)'''` command.
2. View the error log stream output using the BBORBLOG browser. To invoke the browser, go to ISPF option 6 and enter:

   ```
   'BBO.SBBOEXEC(BBORBLOG)' 'BBO.BOSSXXXX format option'
   ```

   Note: In this example, `BBORBLOG` resides in `BBO.SBBOEXEC`.

Results

The browser creates a browser data set named "`userid.stream_name`", which contains the contents of the log stream.

Important: The browser data set is named `userid.stream_name` if the invoking TSO environment includes a TSO PROFILE PREFIX value. If the user ID does not have a TSO PROFILE PREFIX value, then the default output data set name is `stream_name`.

When the browser is executed, it:
1. Allocates a data set called `userid.stream_name`, which overwrites any duplicate data sets.
2. Populates the data set with the contents of the log stream.
3. Puts the user in "browse" mode on the data set.

Important: Each time `BBORBLOG` is invoked a static file is created which overwrites the existing file. In order to refresh the file, it is necessary to re-issue `BBORBLOG`

If the `BBORBLOG` is not in the linklist or in link pack area (LPA), the utility fails with the following error:

```
COMMAND BBORBLOG NOT FOUND 57 *-* ADDRESS TSO "BBORBLOG "strm" "format"" +++ RC(-3) +++
```

Example

There are several valid methods with separate commands to invoke the browser. The following example shows these different methods:

Example: If the BBORBLOG member was in a data set named `BBO.SBBOEXEC`, then you would issue one of the following depending on your chosen format option:

```
ex 'BBO.SBBOEXEC(BBORBLOG)' 'BBO.BOSSXXXX'
ex 'BBO.SBBOEXEC(BBORBLOG)' 'BBO.BOSSXXXX  80'
ex 'BBO.SBBOEXEC(BBORBLOG)' 'BBO.BOSSXXXX  NOFORMAT'
ex 'BBO.SBBOEXEC(BBORBLOG)' 'bbo.bossxxxx NOFORMAT reclen=5200 outdsn=bbos001.stc00123.bborblog'
ex 'BBO.SBBOEXEC(BBORBLOG)' 'bbo.bossxxxx NOFORMAT outdsn=bbos001.stc00456.bborblog'
ex 'BBO.SBBOEXEC(BBORBLOG)' 'bbo.bossxxxx 80 reclen=200'
```

Tip:: It is easier to invoke the browser if you add the target library (in our example, BBO.SBBOEXEC) to the SYSEXEC concatenation of the user logon procedure during the WebSphere Application Server for z/OS installation. If you concatenate the library during installation, you do not have to specify the library containing the browser REXX exec- you only need to specify BBORBLOG.

Error log stream record output

This article provides two samples of error log stream output and explains the various attributes they contain.

There are two error log stream records:
- Server logstream
- CERR of a server.

If you do not want the message tag to be included in the error log output, complete the following actions:
1. In the administrative console, click **Environment > WebSphere variables**.
2. Select the appropriate scope, and then click **New**.
3. Enter ras_error_log_version in the **Name** field and 2 in the **Value** field.
4. Save and synchronize your changes, and then stop, and restart the server.

If you do not want the message tag and the name of the cell, node, and cluster to be included in the error log output, complete the following actions:
1. In the administrative console, click **Environment > WebSphere variables**.
2. Select the appropriate scope, and then click **New**.
3. Enter ras_error_log_version in the **Name** field and 1 in the **Value** field.
4. Save and synchronize your changes, and then stop, and restart the server.

Sample output from a server logstream: The numbers to the left of each sample were added to specify lines. The numbers will not be in the actual output.

```
1| 2008/11/19 22:17:46.325 03 SYSTEM=ATZ3013 CELL=BBOCELL
   NODE=BBONODE  CLUSTER=BBOC001 SERVER=BBOS001 JobName=BBOS001S
2| ASID=0X0053 PID=0X0200025E TID=0X00000045 0XCCDF70CC c=./bbgrjtr.cpp at
3| tag=classlvl... TRAS0028I: The trace output is stored
```

The log stream record output fields from stream BBO.BOSSXXXX are:

Table 85. Description of components. Parts table for a server logstream record output

Component	Description
line 1: 2008/11/19 22:17:46.325 03	Date / timestamp / 2-digit record version number
line 1: SYSTEM=ATZ3013	System name
line 1: CELL=BBOCELL	Cell name
line 1: NODE=BBONODE	Node name
line 1: CLUSTER=BBOC001	Cluster name
line 1: SERVER=BBOS001	Server name

Table 85. Description of components (continued). Parts table for a server logstream record output

Component	Description
line 1: `JobName=BBOS001S`	Jobname
line 2: `ASID=0X0033`	ASID (address space identifier)
line 2: `PID=0X0100003C`	PID (Process ID)
line 2: `TID=0X24F858A0 0X000004`	TID (Thread ID)
line 2: `c=2.1010030`	Request correlation information
line 3: `. /bbooreq.cpp+4437`	File name & line
line 3: `tag=classlvl`	Message tag defined in the classification file
line 3: `BBOU0013W`	Log message number
line 3: `The function...`	Log message
lines 4–5: `make_user_exception... CosNaming::Naming...`	Continuation lines: Continuation of the Log Stream log message

Attention: Each field is delimited by a blank.

Sample output from CERR of a server:

```
1  BossLog: { 0017} 2008/10/01 15:58:25.557 03 SYSTEM=SY1 CELL=BBOCELL NODE=BBONODE  CLUSTER=BBOC001
2  SERVER=BBOS001 JobName=BBOS001S PID=0X0100003C TID=0X24F82920 00000000  c=3.C5D02
3  ./bboiroot.cpp+1195 tag=classlvl... BBOU0012W The function IRootHomeImpl::findHome(
4   const char*)+1195 received CORBA system exception CORBA::INTERNAL.
5   Error code is C9C21200.
```

The CERR job message output fields are:

Table 86. Description of components. Parts table for a CERR record output

Component	Description
line 1: `BossLog: { 0017}`	`BossLog`: {entry number}
line 1: `2000/06/01 15:58:25.557 03`	Date / timestamp / 2-digit record version number
line 1: `SYSTEM=SY1`	System name
line 1: `CELL=BBOCELL`	Cell name
line 1: `NODE=BBONODE`	Node name
line 1: `CLUSTER=BBOC001`	Cluster name

Table 86. Description of components (continued). Parts table for a CERR record output

Component	Description
line 1: `SERVER=BBOS001`	Server name
line 1: `JobName=BBOS001S`	Job name
line 2: `PID=0X0100003C`	PID (Process ID)
line 2: `TID=0X24F82920 00000000`	TID (Thread ID)
line 2: `c=3.C5D02`	Request correlation information
line 3: `. /bboiroot.cpp+1195`	File name & line
line 3: `tag=classlvl`	Message tag defined in the Classification file
line 3: `BBOU0012W`	Log message number
line 3: `The function IRootHomeImpl::find...`	Log message
lines 4-5: `const char*)+1195 received CORBA system exception CORBA::INTERNAL. Error code is C9C21200.`	Continuation lines: Continuation lines of the CERR job message

- Each field is delimited by a blank.
- The CERR format is found in SYSOUT, not the logger.

Saving your BBORBLOG browser output

When you use the BBORBLOG browser, it creates a data set with your user ID, followed by the log stream name. You should rename it if you wish to save your browser output. The contents of the current view of the log stream will remain until the stream reaches its retention date. The next time you invoke the browser; however, the current view of the log stream will be deleted because it uses the same data set name. The previous data will exist in another record, not the current view, until its retention date.

z/OS display command

Use either the WebSphere Application Server administrative console or the z/OS MVS console to accomplish many operations tasks related to WebSphere Application Server for z/OS servers. Entering the z/OS display or modify commands through the MVS console can provide information or perform tasks that are useful for diagnosing problems.

The purpose of the **DISPLAY** command is to display information about the operating system, the jobs and application programs that are running, the processor, devices that are online and offline, central and expanded storage, workload management service policy and mode status, and the time of day.

For example, you could use the command

```
D WLM,APPLENV=*
```

to verify that WLM is defined and available for your applications.

To display the dynamic WLM application environment, use the command:

```
D WLM,DYNAPPL=*
```

You can check the WLM environment for a specific application using the following command:

```
D WLM,APPLENV=appname
```

These commands will return information similar to the following:

```
RESPONSE=SC42
IWM029I 13.09.47 WLM DISPLAY 075
APPLICATION ENVIRONMENT NAME STATE STATE DATA
appname AVAILABLE
ATTRIBUTES:PROC=procname SUBSYSTEM TYPE:CB
```

For more information on the display command see z/OS MVS System Commands, SA22-7627

Hexadecimal conversion of Java error codes

Occasionally, Java will take an WebSphere Application Server for z/OS error code (C9C2xxxx in hexadecimal) and convert it to a very large negative number. If you get a very large negative number, try converting it back to hexadecimal to find the correct code.

To convert the error codes back to hexadecimal you must add 2^{32} to the negative number and convert it into hexadecimal. This can be done using the OMVS command.

```
bc
```

Example: Suppose you get the error code "910022649":

1. Under OMVS, type the command:

   ```
   bc
   ```

2. then type:

   ```
   obase=16
   2^32 - 910022649
   quit
   ```

   The bc program displays C9C22807, which is the hex value that you should look up.

Managing operator message routing

Use the product message routing capabilities to control server traffic flow.

You can route many of the BBO prefixed error messages to specific datasets instead of having them go to SYSLOG, which can create a lot of traffic. This is implemented with the use of two environment variables, ras_default_msg_dd and ras_hardcopy_msg_dd, and the specification of the appropriate DD statement in your JCL start procedure.

The following explains, in more detail how messages get routed.

- WTO messages issued by the Application Server during initialization are sent to hardcopy, but most can be routed to the data set specified by ras_default_msg_dd (see “Log output destinations and characteristics” on page 181).
- The Java audit messages are also sent to hardcopy, but can be routed to the data set specified by ras_hardcopy_msg_dd. (see “Log output destinations and characteristics” on page 181).

- Trace error, service, and fatal messages are sent to the error log specified by the ras_log_logstreamName. Otherwise, they go to CERR (SYSOUT). Some might also go to hardcopy. At the W500104 service level, the ras_log_logstreamName environment variable is not set to the error logstream name in the *was.env* variables.

  To set this environment variable, on the administrative console, click **Environment > WebSphere variables**, select a scope, and click **New**.
- Early error messages go to SYSOUT until the product connects to the log stream. A WTO (BBOO0153I) is issued telling you how many messages went to SYSOUT before you connected to the log stream.
- Starting with z/OS Version 1.13, you can use JES2 DD keywords to segment output using the periodic writing of form-feed characters to the output streams.

  If you are running on z/OS Version 1.12 or earlier, and using JES2, the SEGMENT= parameter can be added to the SYSPRINT and SYSOUT DD cards if you want to segment output using the periodic writing of form-feed characters to the output streams. Form-feed characters are written to the output streams based on the values of the *ras_stderr_ff_interval*, *ras_stdout_ff_interval*, *ras_stderr_ff_line_interval*, and *ras_stdout_ff_line_interval* environment variables. These variables are described in more detail in the topic *Application server custom properties for z/OS* The SEGMENT= parameter is not supported on JES3.

  To set these environment variables, on the administrative console, click **Environment > WebSphere variables**, select a scope, and then click **New**.
- Trace messages are routed to ras_trace_outputLocation.
- System.out.println, System.err.println, STDOUT and cout go to SYSPRINT (see the topic *Redirectiong SYSPRINT and SYSOUT output to an HFS File* for more information.
- STDERR and cerr go to SYSOUT

To use these message routing variables, you must do two things:

1. Add these parameters to the server definitions using the Administrative Console under Environment -> Manage WebSphere Variables:
   - **ras_default_msg_dd =DEFALTDD**
   - **ras_hardcopy_msg_dd =HRDCPYDD**

   You can set these variables for individual control and servant processes, but it is easier to set them in the Environment variables for the entire cell. For the Daemon, you must prefix them with "DAEMON_" and set them at the cell level:
   - **DAEMON_ras_default_msg_dd =DEFALTDD**
   - **DAEMON_ras_hardcopy_msg_dd =HRDCPYDD**
2. Update the procedures in PROCLIB to add these new DD statements:

```
//* Output DDs
//CEEDUMP DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//SYSOUT DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//SYSPRINT DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//DEFALTDD DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//HRDCPYDD DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
```

Note:

- If you specify the new environment variables, but do not specify the DD cards in the procedure, you will not get an error message indicating that the DD cards are missing and the tracing output will not be written anywhere.
- If you try to direct the output for multiple streams to the same DD, such as setting both ras_default_msg_dd and ras_hardcopy_msg_dd to DEFALTDD (or to SYSPRINT) then the allocation will fail and output will be sent to the default location (JOBLOG/SYSLOG).

For example, these DD files are used to segregate the messages and keep almost all of them off the hardcopy console (SYSLOG):

1. JESMSGLG - a few start-up and shut-down messages
2. JESYSMSG - MVS allocation and deallocation messages
3. SYSOUT - a few start-up and shut-down messages
4. SYSPRINT - a few start-up and shut-down messages
5. HRDCPYDD - audit messages that would normally go to SYSLOG
6. DEFALTDD - informational messages that would normally go to SYSLOG

Chapter 17. Configuring the hang detection policy

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.

Before you begin

A common error in Java Platform, Enterprise Edition (Java EE) 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.

Using the hang detection policy, you can specify a time that is too long for a unit of work to complete. The thread monitor checks all managed threads in the system (for example, web container threads and object request broker (ORB) threads) . Unmanaged threads, which are threads created by applications, are not monitored. For more information read “Hung threads in Java Platform, Enterprise Edition applications” on page 271.

About this task

The thread hang detection option is enabled by default. To adjust the hang detection policy values, or to disable hang detection completely:

Procedure

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:

Information	Description
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)

Information	Description
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	600 seconds (ten minutes)

Information	Description
Name	com.ibm.websphere.threadmonitor.false.alarm.threshold

Information	Description
Value	The number of times (T) that false alarms can occur before automatically increasing the threshold. It is possible that a thread that is reported as hung eventually completes its work, resulting in a false alarm. A large number of these events indicates that the threshhold value is too small. The hang detection facility can automatically respond to this situation: For every T false alarms, the threshold T is increased by a factor of 1.5. Set the value to zero (or less) to disable the automatic adjustment.
Default	100

Information	Description
Name	com.ibm.websphere.threadmonitor.dump.java
Value	This property calls the dumpThreads function. Set to true to cause a javacore, a heapdump, and a TDUMP to be created when a hung thread is detected and a WSVR0605W message is printed. The threads section of the javacore can be analyzed to determine what the reported thread and other related threads are doing. See the topic *Dumping threads in server processes using scripting* for a description of the contents of these dumps, and how to disable the creation of one or more of these dumps. Set to an integer value in the range 1 through Integer.MAX_VALUE to cause a javacore to be created when a hung thread is detected and a WSVR0605W message is printed. The integer value indicates the maximum number of javacores that will be created.
Default	false (0)

Information	Description
Name	com.ibm.websphere.threadmonitor.dump.stack
Value	Set to true to cause a stack trace to be printed when a hung thread is detected and a WSVR0605W message is printed.
Default	true

To disable the hang detection option, set the **com.ibm.websphere.threadmonitor.interval** property to less than or equal to zero.

5. Optional: If you want to monitor the activity of threads on which system alarms execute, add the following JVM generic arguments to the server settings.

Information	Description
Name	-Dcom.ibm.websphere.alarmthreadmonitor.generate.javacore
Value	Set to any value to cause a javacore dump to be created when an hung system alarm thread is detected. The threads section of the javacore dump can be analyzed to determine what the reported thread and other related threads are doing.
Default	Unset

Information	Description
Name	com.ibm.websphere.alarmthreadmonitor.checkinterval.millis
Value	The frequency, in milliseconds, at which system alarm threads are interrogated. Set the value to zero to disable system alarm hung thread detection. The maximum interval is 600000 (10 minutes).
Default	10000 (10 seconds)

Information	Description
Name	-Dcom.ibm.websphere.alarmthreadmonitor.threshold.millis
Value	Set to any value integer between 10000 and 600000 (10 minutes). This argument is used to specify the length of time, in milliseconds, that a system alarm thread can be active before it is considered non-responsive. Any system alarm thread that is detected as inactive for longer than this length of time is reported as hung.
Default	10000 (10 seconds)

To add these arguments to the server settings, complete the following actions:

a. Under Server Infrastructure on the server settings page in the administrative console, click **Java and process management** > **Process definition**.
b. Select **Control**.
c. Select **Java virtual machine**.
d. Add the arguments to the JVM generic arguments section.

6. Click **Apply**.
7. Click **OK**.
8. Save the changes. Make sure a file synchronization is performed before restarting the servers.
9. Restart the Application Server for the changes to take effect.

Hung threads in Java Platform, Enterprise Edition applications

WebSphere Application Server monitors thread activity and performs diagnostic actions if one has become inactive.

When WebSphere detects that a thread has been active longer than the time defined by the thread monitor threshold, the application server takes the following actions:

- Logs a warning in the WebSphere Application Server log that indicates the name of the thread that is hung and how long it has already been active. The following message is written to the log:

  ```
  WSVR0605W: Thread threadname has been active for
  hangtime and may be hung.  There are totalthreads
  threads in total in the server that may be hung.
  ```

  where: *threadname* is the name that appears in a JVM thread dump, *hangtime* gives an approximation of how long the thread has been active and *totalthreads* gives an overall assessment of the system threads.
- Issues a Java Management Extensions (JMX) notification. This notification enables third-party tools to catch the event and take appropriate action, such as triggering a JVM thread dump of the server, or issuing an electronic page or email. The following JMX notification events are defined in the com.ibm.websphere.management.NotificationConstants class:
  - TYPE_THREAD_MONITOR_THREAD_HUNG This event is triggered by the detection of a (potentially) hung thread.
  - TYPE_THREAD_MONITOR_THREAD_CLEAR This event is triggered if a thread that was previously reported as hung completes its work. Consult the section on false alarms for more information.
- Triggers changes in the performance monitoring infrastructure (PMI) data counters. These PMI data counters are used by various tools, such as the Tivoli Performance Viewer, to provide a performance analysis.
- Triggers changes in the performance monitoring infrastructure (PMI) data counters. These PMI data counters are used by various tools, such as the Tivoli Performance Viewer, to provide a performance analysis.

For additional information about performance monitoring and Tivoli Performance Viewer, see the chapter Monitoring performance with Tivoli Performance Viewer in the *Tuning guide* PDF book

False Alarms

If the work actually completes, a second set of messages, notifications and PMI events is produced to identify the false alarm. The following message is written to the log:

```
WSVR0606W: Thread threadname was previously reported to be
hung but has completed. It was active for approximately hangtime.
There are totalthreads threads in total in the server that still
may be hung.
```

where *threadname* is the name that appears in a JVM thread dump, *hangtime* gives an approximation of how long the thread has been active and *totalthreads* gives an overall assessment of the system threads.

Automatic adjustment of the hang time threshold

If the thread monitor determines that too many false alarms are issued (determined by the number of pairs of hang and clear messages), it can automatically adjust the threshold. When this adjustment occurs, the following message is written to the log:

```
WSVR0607W: Too many thread hangs have been falsely reported.  The hang
threshold is now being set to thresholdtime.
```

where: *thresholdtime* is the time (in seconds) in which a thread can be active before it is considered hung.

You can prevent WebSphere Application Server from automatically adjusting the hang time threshold. See Chapter 17, “Configuring the hang detection policy,” on page 269

System Alarms

An application server monitors the activity of threads on which system alarms execute. When a system alarm thread has been active longer than the time defined by the alarm thread monitor threshold, the application server logs the following warning in the system log. This message indicates the name of the thread that is not responding, the length of time that the thread has already been active, and the exception stack of the thread, which identifies the system component.

```
UTLS0008W: The alarm thread threadname has been active for n
   milliseconds and may be hung. totalthreadsthreadstack
```

In this message, *threadname* is the name that appears in a JVM thread dump, *n* is approximately how long the thread was active, *totalthreads* is an overall assessment of the system threads, and *threadstack* is the exception stack of the thread.

If the alarm work eventually completes, the following message is written to the system log. This message indicates thread that produced the false alarm.

```
UTLS0009W: Alarm Thread threadname was previously reported to be hung but has
   completed.  It was active for approximately n milliseconds.
```

In this message, *threadname* is the name that appears in a JVM thread dump, and *n* is approximately how long the thread was active.

Typically, system alarms do not process heavy loads because such activity might slow the processing of later system alarms, which in turn might impact server behavior. The UTLS0008W message is intended to help IBM Support personnel investigate problems potentially caused by system alarm behavior.

All of the system alarms share a common alarm thread pool. The properties which govern the monitoring of this thread pool can be tuned using the administrative console. You can reduce the frequency at which WebSphere generates alarm hung thread messages by adjusting the alarm thread monitor check interval or threshold. See the topic *Configuring the hang detection policy* for a description of how to change these settings.

Example: Adjusting the thread monitor to affect server hang detection

The hang detection policy affects how the application server responds to a thread that is not being processed correctly.

You can adjust the thread monitor settings by using the wsadmin scripting interface. These changes take effect immediately, but do not persist to the server configuration, and are lost when the server is restarted. The following script provides an example of how to adjust the properties for the thread monitor using the wsadmin tool:

```
# Read in the interval, threshold, false alarm from the command line
set interval [lindex $argv 0]
set threshold [lindex $argv 1]
set adjustment [lindex $argv 2]

# Get the object name of the server you want to change the values on
set server [$AdminControl completeObjectName "type=Server,*"]

# Read in the interval and print to the console
set i [$AdminControl getAttribute $server threadMonitorInterval]

# Read in the threshold and print to the console
set t [$AdminControl getAttribute $server threadMonitorThreshold]

# Read in the false alarm adjustment threshold and print to the console
set a [$AdminControl getAttribute $server threadMonitorAdjustmentThreshold]

# Set the new values using the command line parameters
$AdminControl setAttribute $server threadMonitorInterval  ${interval}

$AdminControl setAttribute $server threadMonitorThreshold ${threshold}

$AdminControl setAttribute $server threadMonitorAdjustmentThreshold ${adjustment}
```

Chapter 18. Automation and recovery scenarios and guidelines

The following section provides information on how to monitor and recover WebSphere Application Server for z/OS and the subsystems it uses.

It provides startup, shutdown, and recovery procedures and scenarios. It also tells you how to determine if the subsystems are up or down, and tells you where to find more information.

APPC automation and recovery scenarios

This page contains scenarios and information on the automation and recovery of APPC.

Table 87. Scenarios and information on the automation and recovery of APPC. Use the following table to diagnose various APPC problems.

Task	APPC automation and recovery scenarios
Startup	APPC should be started before WebSphere Application Server for z/OS. In theory, WebSphere Application Server for z/OS *could* be started before APPC, but only as long as no objects get dispatched in containers that have an IMS APPC LRMI associated with them. If APPC is not up before WebSphere Application Server for z/OS, and you want to use an APPC connector to talk to IMS, then you will have no connectivity. APPC/MVS does not have to be up for CICS. APPC does not have to be started after VTAM®.
Shutdown	Reverse the startup procedure. Shutdown WebSphere Application Server for z/OS, APPC, then VTAM.
Handling in-flight or indoubt transactions if there is a failure	If you are using APPC for communications and it fails, do the following: 1. Shutdown all servers with APPC connectivity. 2. Restart APPC (if it totally failed). 3. Restart the WebSphere Application Server for z/OS server. **Note:** APPC will resynchronize itself. If your transaction is indoubt, IMS waits until you restart APPC. IMS relies on RRS for recovery. RRS will resolve transactions that are in doubt by handshaking with every subsystem it was communicating with before it went down. If you are using CICS, note that CICS has its own coordinator.
How to determine if APPC is running	Issue the `DISPLAY APPC,LU,ALL` command. If APPC is not active, it will say so. In addition, the status of the logical units used by WebSphere Application Server for z/OS and/or IMS should be active or no APPC work will be successful.
What happens to WebSphere for z/OS if APPC goes down?	Any objects attempting to use the IMS APPC PAA will not work. The server region running on behalf of the container attempting to use APPC will likely get a `C9C24C05` error, indicating that an APPC ALLOCATE request was attempted and failed. Additional APPC error diagnostic information that helps to pinpoint the APPC problem is contained in the logs associated with this region.
What happens to other subsystems if APPC goes down?	Not applicable
Where to find more information	• *z/OS MVS Planning: Operations* • *z/OS MVS Planning: APPC/MVS Management* • *z/OS MVS Programming: Resource Recovery*

WLM automation and recovery scenarios

This table provides scenarios for automation and recovery of WLM enabled systems.

Table 88. WLM automation and recovery scenarios. The following tables lists tasks and the WLM automation and recovery scenarios for the task.

Task	WLM automation and recovery scenarios
Startup	WLM is automatically started by z/OS when you IPL your system. You do not have to start it.
Shutdown	You cannot shutdown WLM.
Handling in-flight and indoubt transactions if there is a failure	Not applicable
How to determine if WLM is running	Not applicable
What happens to the product if WLM goes down?	Not applicable
What happens to other subsystems if WLM goes down?	Not applicable

How to handle a catastrophic failure of the servants	Following a catastrophic failure of the servants, you can use one of the following resume commands, depending on your application environment type:
	• Static application environment:
	`v wlm,applenv=applenvname, resume`
	• Dynamic application environment:
	`v wlm,dynappl=applenvname, resume,options`

Where to find more information	• *z/OS MVS Planning: Workload Management* • *z/OS MVS Programming: Workload Management Services*

RACF automation and recovery scenarios

This table provides scenarios for RACF automation and recovery.

Table 89. Scenarios for RACF automation and recovery. RACF program provides improved security for an installation data. RACF protects your vital system resources and controls what users can do on the operating system.

Task	RACF automation and recovery scenarios
Startup	If it is installed, RACF is started as a part of IPL.
Shutdown	RACF is not shutdown.
Handling in-flight and indoubt transactions if there is a failure	Not applicable
How to determine if RACF is running	Use the RACF SETROPTS command to display the status of RACF.
What happens to WebSphere for z/OS if RACF goes down?	RACF goes into fail safe mode. This means that for every resource that is accessed, the operator is asked to verify if it is okay. In general, the system is IPLed if this occurs.
What happens to other subsystems if RACF goes down?	It depends on the subsystem and how RACF fails.
Where to find more information	• *z/OS Security Server RACF System Programmer's Guide* • *z/OS Security Server RACF Security Administrator's Guide*

RRS automation and recovery scenarios

Use this table for RRS automation and recovery scenarios.

Table 90. RRS automation and recovery scenarios. Although RRS restart and resource manager restart are two separate items, they must be considered together because in many instances RRS restart issues have a major effect on resource manager restart issues. For additional information, read about the RRS restart and recovery section in the Systems Programmer's Guide to Resource Recovery Services (RRS) book.

Task	RRS automation and recovery scenarios
Startup	Ensure System Logger has been started before RRS. **Note:** RRS will display error messages indicating that System Logger must be started first if you try to start RRS without starting System Logger.Ensure RRS is started before WebSphere for z/OS. RRS does not start by itself. RRS will start automatically only if it was registered with the Automatic Restart Manager (ARM) and if ARM is running. To start RRS, issue the `start` command: `start atrrrs,sub=master` **Note:** RRS doesn't restart itself if you issue the cancel command, so you need to restart it manually if it was canceled or if ARM isn't running.
Shutdown	Shutdown RRS in the reverse order that you started RRS. Shutdown WebSphere Application Server for z/OS, then RRS, followed by System Logger. There is no controlled way to bring down RRS. The best approach is: 1. Quiesce WebSphere Application Server for z/OS. 2. Shutdown WebSphere Application Server for z/OS. 3. Cancel RRS. **Note:** You may want to bring down the DB2 you are using for WebSphere Application Server for z/OS before canceling RRS. To cancel RRS, issue the command: `setrrs cancel`
Handling in-flight and indoubt transactions if there is a failure	Refer to the RRS system management panels to display in-flight and resolve indoubt transactions. You can display the resource managers on the RM panels in RRS, display all units of recovery (UR), filter the URs, and then resolve the indoubts. You cannot resolve in-flights. You can display all RRS-managed transactions. If you are using the IMS Connector for Java, this process applies only if IMS Connector for Java, IMS Connect, and the IMS subsystem are configured locally on the same z/OS system image on which the WebSphere for z/OS J2EE server runs. The local configuration is the only configuration in which IMS Connector for Java runs as an RRS-transactional connector.
How to determine if RRS is running	Use the display command: `d a,atrrs` `atrrs` is the name of the default RRS procedure shipped with WebSphere Application Server for z/OS. Use the procedure name that you use to start RRS. The address space comes from the procedure.
What happens to WebSphere for z/OS if RRS goes down?	RRS is a required subsystem, so WebSphere Application Server for z/OS will not run without it. If RRS goes down, WebSphere Application Server for z/OS will get fatal errors. You need to get RRS started, then restart WebSphere Application Server for z/OS.

Table 90. RRS automation and recovery scenarios (continued). Although RRS restart and resource manager restart are two separate items, they must be considered together because in many instances RRS restart issues have a major effect on resource manager restart issues. For additional information, read about the RRS restart and recovery section in the Systems Programmer's Guide to Resource Recovery Services (RRS) book.

Task	RRS automation and recovery scenarios
What happens to other subsystems if RRS goes down?	RRS is the WebSphere Application Server for z/OS transaction monitor. If you cancel RRS, you will have problems with any subsystems using it (for example, WebSphere Application Server for z/OS, DB2, IMS). You should understand the implications before you cancel RRS.
Where to find more information	*z/OS MVS Programming: Resource Recovery*

UNIX System Services automation and recovery scenarios

This table provides automation and recovery scenarios for using UNIX System Services.

Table 91. Automation and recovery scenarios for using UNIX System Services. The MVS operator can use the DISPLAY command to obtain the UNIX System Services status, for example, active or terminating, shutting down or restarting.

Task	UNIX System Services automation and recovery scenarios
Startup	UNIX System Services is a permanent component of MVS and is started automatically at IPL time.
Shutdown	You can use the MODIFY command to control z/OS UNIX System Services and to terminate a z/OS UNIX process or thread. You can also use it to shut down z/OS UNIX initiators and to request a SYSMDUMP for a process. For more information on the MODIFY command see z/OS MVS System Commands, SA22-7627.
Handling in-flight or indoubt transactions if there is a failure	The only data that could be considered transactional in nature is data stored in the HFS.
How to determine if UNIX System Services is running	UNIX System Services is always available as long as the system is up and running.
What happens to WebSphere for z/OS if UNIX System Services goes down?	If UNIX System Services fails, the system must be re-IPLed, or you can use the MODIFY OMVS command to recycle z/OS UNIX System Services. This is an alternative to re-IPLing the system in order to reinitialize the z/OS UNIX System Services environment. This command should be used only on a limited basis when complete reinitialization and reconfiguration are required. Prior to issuing MODIFY OMVS to initiate a shutdown, you should review the information about shutdown in z/OS UNIX System Services Planning. For more information on the MODIFY command see z/OS MVS System Commands, SA22-7627.

Table 91. Automation and recovery scenarios for using UNIX System Services (continued). The MVS operator can use the DISPLAY command to obtain the UNIX System Services status, for example, active or terminating, shutting down or restarting.

Task	UNIX System Services automation and recovery scenarios
What happens to other subsystems if UNIX System Services goes down?	If UNIX System Services fails, then other subsystems will be adversely affected until the problem is resolved. The system must be re-IPLed, or you can use the MODIFY OMVS command to recycle z/OS UNIX System Services. This is an alternative to re-IPLing the system in order to reinitialize the z/OS UNIX System Services environment. This command should be used only on a limited basis when complete reinitialization and reconfiguration are required. Prior to issuing MODIFY OMVS to initiate a shutdown, you should review the information about shutdown in z/OS UNIX System Services Planning. For more information on the MODIFY command see z/OS MVS System Commands, SA22-7627.
Where to find more information	*z/OS UNIX System Services Planning*

TCP/IP automation and recovery scenarios

This table provides automation and recovery scenarios for TCP/IP configurations.

Table 92. Automation and recovery scenarios for TCP/IP configurations. To prepare TCP/IP on z/OS, read about the information in the Preparing TCP/IP on z/OS topic.

Task	TCP/IP automation and recovery scenarios
Startup	TCP/IP must be up before starting WebSphere Application Server for z/OS.
Shutdown	Shutdown WebSphere Application Server for z/OS before shutting down TCP/IP.
Handling inflight or indoubt transactions if there is a failure	Methods in flight might have their transactions rolled back when the attempt to send a response to the method fails. Other transactions might wait for a timeout.
How to determine if TCP/IP is running	Use the display command looking for the TCP/IP procedure.
What happens to WebSphere Application Server for z/OS if TCP/IP goes down?	Restart TPC/IP. After TCP/IP is restarted, you also must restart WebSphere Application Server. If you are using another product to supplement WebSphere Application Server for z/OS, you might need to restart the application server for the other programs to re-initialize the TCP/IP settings.
What happens to other subsystems if TCP/IP goes down?	If TCP/IP goes down, sessions break and transactions react as described above. **Note:** If TCP/IP goes down, LDAP functionality might be affected. You might need to recycle LDAP and restart WebSphere Application Server for z/OS if this occurs.

DB2 automation and recovery scenarios

This article provides scenarios for automation and recovery of DB2 resources.

Table 93. DB2 automation and recovery scenarios. DB2 for z/OS is the database of choice for critical data for many enterprises. It is becoming more important to protect this data in case of disaster and to be able to restart with a consistent copy of the DB2 data as quick as possible and with minimal losses.

Task	DB2 automation and recovery scenarios
Startup	DB2 is started after RRS but before LDAP, NFS, and WebSphere Application Server for z/OS.
Shutdown	Reverse of startup sequence.

Handling in-flight or indoubt transactions if there is a failure	Use the RRS panels to resolve. See *z/OS MVS Programming: Resource Recovery*. The RRS panels are the preferred way to resolve DB2 indoubts because they allow you to view all resource managers that have an interest in the transaction. However, you can also use DB2 to resolve indoubts. You can issue the command:
	`DISPLAY THREAD(*) TYPE(INDOUBT)`
	
	to display DB2 information about the indoubt threads it knows about (if there are too many, you can go into S.LOG to view the information). This display will give you a DB2 identifier called a "nid". Copy the nid and paste it into this command:
	`-RECOVER INDOUBT (RRSAF) ACTION(COMMIT)`
	`NID(B1D379D17ED6CF900000009401010000)`
	
	where the `nid` is the one that you cut from the display command. You can issue this command to roll back the transaction:
	`-RECOVER INDOUBT (RRSAF) ACTION(ABORT)`
	`NID(B1D379D17ED6CF900000009401010000)`

How to determine if DB2 is running	Use the display command to display the DB2 address space.
What happens to WebSphere Application Server for z/OS if DB2 goes down?	WebSphere Application Server for z/OS continues to run. WebSphere Application Server for z/OS does not require restarting in this scenario.
What happens to other subsystems if DB2 goes down?	Not applicable
Where to find more information	See the DB2 books at the following Internet location:http://www.ibm.com/servers/eserver/zseries/zos/

CICS automation and recovery scenarios

This article explains different scenarios for CICS automation and recovery.

Table 94. Scenarios for CICS automation and recovery. The MVS operator can use the DISPLAY command to obtain the UNIX System Services status, for example, active or terminating, shutting down or restarting.

Task	CICS automation and recovery scenarios
Startup	CICS and any required CICS products, such as CICS Transaction Gateway, need to be properly installed, initialized, and started before any work flows to a CICS-enabled WebSphere Application Server controller.
Shutdown	Shutdown the WebSphere Application Server controller that uses CICS as a backing store, then shut down the CICS service.

Table 94. Scenarios for CICS automation and recovery (continued). The MVS operator can use the DISPLAY command to obtain the UNIX System Services status, for example, active or terminating, shutting down or restarting.

Task	CICS automation and recovery scenarios
Handling in-flight or indoubt transactions if there is a failure	If there is an error during processing, both CICS and WebSphere Application Server for z/OS rely on the underlying RRS subsystem to handle all rollback notifications to the registered interests. In the case of inflight transactions, RRS will notify all participants that a rollback is required, and normal rollback processing will occur in each registered party. In the case of indoubt transactions, it may be necessary to recycle the WebSphere Application Server for z/OS Application Control/Server region to release any pending transaction in CICS.
How to determine if CICS is running	This is installation dependent.
Troubleshooting the failure of CICS web transactions that trigger the error **WSIFIOException**	Failure occurs because the CICS resource adapter cannot access the CTG*.so files, which the adapter requires for running the Web Services Invocation Framework (WSIF). Solve the problem with the following steps: 1. Copy the CTG*.so files into the /lib directory of your Application Server installation. For example, the/`lib` directory might have the following path: `WebSphere/V7R0/AppServer1/lib/`. 2. Restart the CICS web transaction service over WSIF.
What happens to CICS if WebSphere Application Server for z/OS goes down?	Should WebSphere Application Server for z/OS happen to go down, one of two situations could occur: 1. If WebSphere Application Server for z/OS and CICS are currently engaged in a unit of work, then RRS processing as described above would occur and it may be necessary to recycle the application control server regions to release pending transactional work in CICS. 2. If WebSphere Application Server for z/OS and CICS are not currently engaged in a unit of work, CICS is not affected.
What happens to other subsystems if CICS goes down?	Not applicable
Where to find more information	*CICS Operations and Utilities Guide*

IMS automation and recovery scenarios

This article provides scenarios for IMS automation and recovery.

Table 95. Scenarios for IMS automation and recovery. When a failure occurs, you must perform the correct recovery and restart procedures to ensure system integrity.

Task	IMS automation and recovery scenarios
Startup	IMS and any required IMS products, such as IMS Connect, need to be properly installed, initialized, and started before any work flows to an IMS-enabled WebSphere for z/OS application control server region are run.
Shutdown	Shutdown the WebSphere Application Server for z/OS application controller that uses IMS as a backing store, then shutdown the IMS service

Table 95. Scenarios for IMS automation and recovery (continued). When a failure occurs, you must perform the correct recovery and restart procedures to ensure system integrity.

Task	IMS automation and recovery scenarios
Handling in-flight or indoubt transactions if there is a failure	If there is an error during processing, both IMS and WebSphere Application Server for z/OS rely on the underlying RRS subsystem to handle all rollback notifications to the registered interests. In the case of inflight transactions, RRS will notify all participants that a rollback is required and normal rollback processing will occur in each registered party. In the case of indoubt transactions, it may be necessary to recycle the WebSphere Application Server for z/OS Application Control/Server region to release any pending transaction in the IMS MPRs.
How to determine if IMS is running	This is installation-dependent.
What happens to IMS if WebSphere Application Server for z/OS goes down?	Should WebSphere Application Server for z/OS happen to go down, one of two situations could occur: 1. If WebSphere Application Server for z/OS and IMS are currently engaged in a unit of work, then RRS processing as described above would occur and it may be necessary to recycle the application control server regions to release pending transactional work in the IMS MPR. 2. If WebSphere Application Server for z/OS and IMS are not currently engaged in a unit of work, IMS is not affected.
What happens to other subsystems if IMS goes down?	Not applicable
Where to find more information	*IMS/ESA® Operator's Reference*

WebSphere Application Server for z/OS (Daemon) automation and recovery scenarios

This article provides some scenarios for automation and recovery of WebSphere Application Server for z/OS.

Table 96. Scenarios for automation and recovery of WebSphere Application Server for z/OS. To support WebSphere Application Server for z/OS to restart on an alternate system, a few prerequisites must be installed on every system (your original system as well as any systems intended for recovery) before reconfiguring the ARM policies to enable peer restart and recovery. For more information, read about the Setting up peer restart and recovery topic.

Task	WebSphere Application Server for z/OS (daemon) automation and recovery scenarios
Startup	See the instructions described in the where to perform WebSphere Application Server operations documentation.
Shutdown	See the instructions described in the where to perform WebSphere Application Server operations documentation.
Handling inflight or indoubt transactions if there is a failure	The daemon is a location agent. If the daemon fails during the course of a transaction, locate requests to the daemon can fail. These request failures might be surfaced by the client ORB. If the client is a WebSphere Application Server for z/OS client running in a sysplex, the locate request can be routed to another available daemon in the sysplex, if present.
How to determine if the daemon is running	Use the MVS display command.
What happens to WebSphere Application Server for z/OS if the daemon goes down?	If the daemon goes down, all WebSphere Application Server for z/OS servers started on the same system also are terminated.

Table 96. Scenarios for automation and recovery of WebSphere Application Server for z/OS (continued). To support WebSphere Application Server for z/OS to restart on an alternate system, a few prerequisites must be installed on every system (your original system as well as any systems intended for recovery) before reconfiguring the ARM policies to enable peer restart and recovery. For more information, read about the Setting up peer restart and recovery topic.

Task	WebSphere Application Server for z/OS (daemon) automation and recovery scenarios
What happens to other subsystems if the daemon goes down?	Other subsystems continue to work fine. As a general rule, if the servers on multiple systems are defined in cluster and there is another one in the sysplex, if the daemon goes down , clients are not affected.

Web server (servlet) automation and recovery scenarios

This table provides scenarios for web server (or servlet) automation and recovery.

Table 97. Web server (or servlet) scenarios.. Displays web server (or servlet) scenarios for automation and recovery.

Task	WebServer automation and recovery scenarios
Startup	Web servers have a relationship with WebSphere Application Server for z/OS only in the sense that a client application program that is written to use WebSphere Application Server for z/OS facilities may be written as a servlet. Any implications for ordering of startup will be introduced by the applications. You probably want to have the WebSphere Application Server for z/OS servers up and ready before starting the client application that the web server is hosting.
Shutdown	There are no dependencies from the product code. Similar to most applications, you may want to quiesce the clients prior to taking down the target WebSphere Application Server for z/OS servers. Shut down the web server to stop the port of entry.
Handling in-flight or indoubt transactions if there is a failure	Since a web server is stateless, there are no in-flight or indoubt transactions.
How to determine if a web server is running	Use the z/OS display commands and viewer tools such as SDSF or the administrative console, to monitor the web server.
What happens to WebSphere Application Server for z/OS if the web server goes down?	WebSphere Application Server for z/OS can be enhanced when combined with an IBM HTTP Web Server for more robust load balancing and failover.
What happens to other subsystems if a Web Server goes down?	There is no effect on other subsystems.
Where to find more information	*z/OS HTTP Server Planning, Installing, and Using* or the documentation for your particular web server.

Chapter 19. 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.

About this task

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 website.

Procedure

1. Select the appropriate tool for the task. For more information on the capacities of the supplied troubleshooting tools, see the relevant articles in this section.
2. Run the tool as described in the relevant article.
3. Contact IBM Support for assistance in deciphering the output of the tool. For current information available from IBM Support on known problems and their resolution, see the IBM Support page. IBM Support has documents that can save you time gathering information needed to resolve this problem. For the last minute updates, limitations, and known problems, see the release notes. Before opening a PMR, see the Must gather page.
4. Use the IBM Support Assistant to help find and use various IBM Support resources, such as updated documentation and problem determination tools.

First failure data capture (FFDC)

When a failure occurs during product runtime processing, the (FFDC) feature instantly collects information about the events and errors that lead up to the failure. The captured data can then be used to analyze the problem. After a maximum number of days, these files are automatically deleted from your system.

After the information, which is uniquely identified for the servant region that produced the exception, is collected, and saved in a log file, FFDC returns control to the affected engines.

By default, a FFDC log file is automatically purged seven days it is created. You can configure the amount of days between purges if you are concerned about the amount of space that the FFDC log files are using.

Two FFDC implementations are provided in the product:

- The WebSphere FFDC, which is the legacy FFDC implementation. This FFDC can only be used in WebSphere products.
- The IBM FFDC, which is a more componentized, and more generic implementation that depends solely on the JDK. This FFDC can be used in client processes and by non-WebSphere products, because it is pluggable with non-WebSphere data collectors, formatters, providers, and listeners.

Both of these FFDC implementations support the OnDirProvider type functionality that is configurable using the com.ibm.ffdc.log Java environment variable. The OnDirProvider functionality includes a built-in provider that stores incidents as separate files in a directory, along with a separate summary file. The com.ibm.ffdc.log Java environment variable can be set to the following values:

- ***<file_name>***, where *file_name* is either be the name of a single file or a directory path.
  - If *file_name* exists and is the name of a single file, all of the incident and summary reporting information that FFDC collects is appended into that file.

  - If *file_name* exists and is a directory path, whenever an incident occurs, a new file is created in that directory and all of the incident and summary reporting information for that incident is written into this newly created file. The incident is also added to the summary report in this directory.
  - If *file_name* ends in a file separator (\ or /). but a file with the specified name does not exist, a directory called *file_name* is created. Then, whenever an incident occurs, a new file is created in that directory and all of the incident and summary reporting information for that incident is written into this newly created file. The incident is also added to the summary report in this directory.
  - If *file_name* does not end in a file separator (\ or /). and a file with the specified name does not exist, a single file is created and given the specified name. All of the incident and summary reporting information that FFDC collects is then appended into that file.
- `System.out`, which appends the incidents and summary report information to the stdout output stream.
- `System.err`, which append the incidents and summary report information to the stderr output stream. System.err is the default value for the com.ibm.ffdc.log Java environment variable.
- `Suppress`, which causes all FFDC collected information to be discarded.

Specifying a value for the com.ibm.ffdc.log Java environment variable is the only configuration change that you need to make to exploit the OnDirProvider functionality. The new FFDC also provides mechanisms to overly choose this provider or to use your own WebSphere provider.

Starting in WebSphere Application Server V8, this same variable can be used to redirect the FFDC incidents and summary created in the product. The only acceptable value for that, however, is a directory. So it is recommended that, if this variable is used in a WebSphere server environment, that the specification end in a File separator (/ or \\) to avoid conflicts.

gotcha: If the default setting for automatic purging of FFDC information is too long for your environment, see the topic Configuring first failure data capture log file purges for a description of how you can modify the length of time that the FFDC information is retained on your system.

Configuring first failure data capture log file purges

The first failure data capture (FFDC) log file saves information that is generated from a processing failure. These files are deleted after a maximum number of days has passed. The captured data is saved in a log file for analyzing the problem.

Before you begin

FFDC is intended primarily for use by IBM Support. FFDC instantly collects events and errors that occur during the product run time. The information is captured as it occurs and is written to a log file that can be analyzed by IBM Support. The data is uniquely identified for the servant region that produced the exception.

The log file purges in seven days by default. If you are concerned about the amount of disk space used by the FFDC log files, you can configure the days between purges.

The FFDC configuration properties files are located in the properties directory under the Application Server product installation. You must set the ExceptionFileMaximumAge property to the same value in all three files: `ffdcRun.properties`, `ffdcStart.properties`, and `ffdcStop.properties`. You can set the ExceptionFileMaximumAge property to configure the days between purging the FFDC log files. The value of the ExceptionFileMaximumAge property must be a positive number. The FFDC feature does not affect the performance of the Application Server product.

About this task

Perform the following steps to configure the number of days between the FFDC log file purges. The value is in days.

Procedure

1. Open the `ffdcRun.properties` file.

   The file is located in the *app_server_root*/`properties` directory.
2. Change the value for the *ExceptionFileMaximumAge* property to the number of days between the FFDC log file purges. The value of the *ExceptionFileMaximumAge* property must be a positive number. The default is seven days. For example, `ExceptionFileMaximumAge = 3` sets the default time to three days. The FFDC log file is purged after three days.
3. Save the `ffdcRun.properties` file and exit.
4. Repeat the previous steps to modify the `ffdcStart.properties` and `ffdcStop.properties` files.

Results

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

The FFDC file management function removes the FFDC log files that have reached the maximum age and generates a message in the SYSOUT of the server job log.

Chapter 20. Working with Diagnostic Providers

Diagnostic Providers enable you to query the startup configuration, current configuration, and current state of a diagnostic domain. In addition, Diagnostic Providers can also provide access to any self diagnostic tests that are available from a diagnostic domain.

About this task

The Diagnostic Provider Utility is a simple front end in the administration console that presents the available set of Diagnostic Providers and enables you to work with them.

Procedure

Learn about Diagnostic Providers

Diagnostic Providers

Diagnostic Providers are a quick method for viewing configuration and the current state of individual components within an application server environment.

WebSphere Application Server components can be considered as being divisible into *diagnostic domains*. A diagnostic domain refers to a set of classes within the component that share a set of diagnostics. Some larger components might have multiple diagnostic domains. For example, the Connection Manager logically consists of multiple data sources and connection factories that each have separate diagnostic domains.

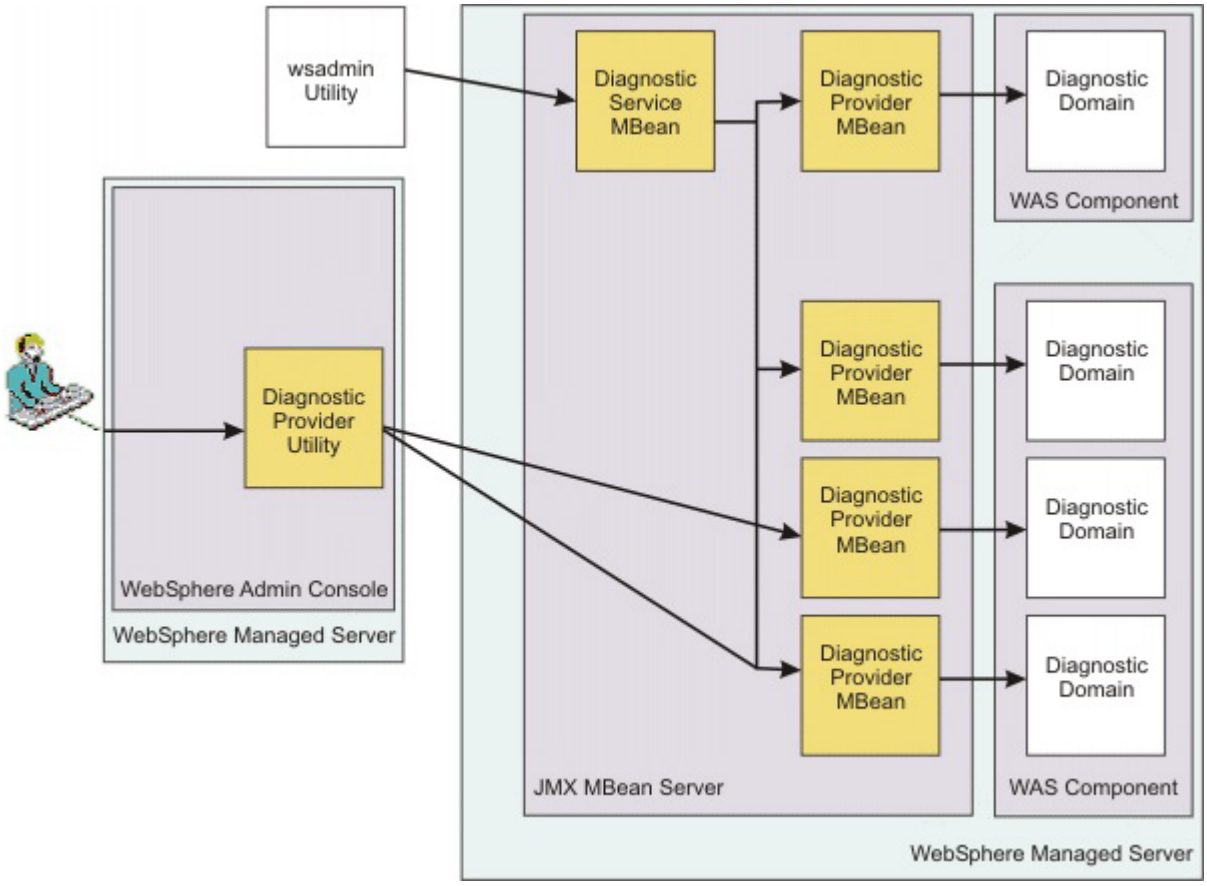


Figure 7. Diagnostic providers. This image shows the relationships between the parts that make up the Diagnostic Provider (DP) utility.

A single diagnostic domain receives its diagnostic services from a Diagnostic Provider MBean. The Diagnostic Provider MBean enables you to query the startup configuration, current configuration, and current state of the diagnostic domain. In addition, Diagnostic Provider MBeans can also provide access to any self diagnostic tests that are available from the diagnostic domain. Some characteristics of Diagnostic Provider MBeans include:

- Diagnostic Provider MBeans are Java Management Extensions (JMX) MBeans
- Diagnostic Provider MBeans all implement a DiagnosticProvider interface which includes methods for configuration dumps, state dumps, and self diagnostic tests
- Diagnostic Provider MBeans provide a way to expose information about running components so administrators can more easily debug problems related to those components. As with other MBeans running in WebSphere Application Server, they can be accessed from JMX client code, or through the *wsadmin* tool.

Diagnostic Provider MBeans are efficient at delivering Java object representations of configuration, state, and self test information. This is good for when programs interact. For human users to access the information, WebSphere Application Server provides a set of facilities to extend the value of Diagnostic Provider MBeans.

The Diagnostic Service MBean

provides methods to convert Diagnostic Provider MBean output into human readable formats. The Diagnostic Service MBean also provides some methods to facilitate looking up the Diagnostic Provider MBeans on the same server as the Diagnostic Service MBean. For administrators that want to access diagnostic data from a command line, the wsadmin tool can be used directly with the Diagnostic Service MBean to get formatted results

The Diagnostic Provider utility
a set of panels included in the WebSphere Application Server administration console through which administrators can interact with Diagnostic Provider MBeans. The Diagnostic Provider utility is a simple front end in the administration console that presents the available set of Diagnostic Provider MBeans present on each managed server, and provides a means to execute and view the results of configuration dumps, state dumps, and diagnostic self tests.

.

The purpose of Diagnostic Providers

Diagnostic Providers give you more information for quickly discovering and diagnosing system problems. The following scenario contrasts the experience of an administrator working with a component that does not have a Diagnostic Provider to one that does.

When the administrator works with a component that is without a Diagnostic Provider, the events are as follows:
1. A log entry indicates that a particular component is experiencing a problem.
2. The system administrator sees the log entry through the runtime messages panel.
3. The system administrator cannot tell what is wrong, so calls IBM support for assistance, with a potentially ill-defined problem.

When the administrator works with a component with a Diagnostic Provider, and the Diagnostic Provider ID is registered with the component's logger, the situation changes as follows:
1. A log entry that contains a Diagnostic Provider ID (DPID) indicates that something has gone wrong in a specific component.
2. The system administrator sees the log entry through the runtime messages panel.
3. The administrator clicks a button on the runtime message panel to execute a state dump or a configuration dump, or to be taken to the list of component self tests.
4. From the self test, the administrator is warned that the component is configured in a way that could lead to poor performance or failures.

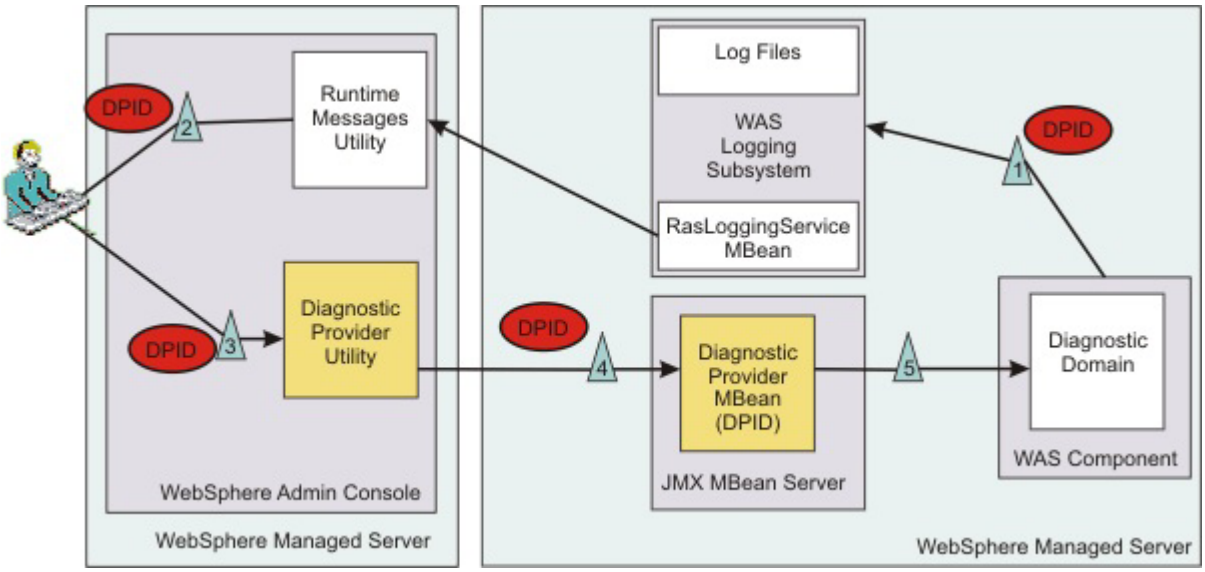
Furthermore, when the administrator works with a component with a Diagnostic Provider, and the Diagnostic Provider ID is **not** registered with the component's logger, the situation might unfold like this:
1. A log entry which doesn’t contain a DPID indicates that something has gone wrong in a component.
2. The system administrator sees the log entry through the runtime messages panel.
3. The system administrator uses the administrative console to navigate through the available set of Diagnostic Providers and selects one that sounds appropriate.
4. He runs a configuration dump, a state dump, or a self diagnostic test against the Diagnostic Provider to collect information about the component.
5. From the state dump, the administrator is able to notice that the component state is not what would be expected for its workload.
6. The administrator works with the test team to determine which of the flows is causing the state of the component to diverge from what is expected (as evidenced by repeated execution of the state dump).

Diagnostic Provider IDs

A Diagnostic Provider ID (DPID) is the unique address of a Diagnostic Provider MBean. Components that have associated Diagnostic Provider MBeans can include the DPID in their log entries.

Diagnostic Provider IDs are implemented in WebSphere Application Server as Java Management Extensions (JMX) *MBean ObjectNames*, and can be used at JMX MBean servers to look up Diagnostic Provider MBeans.

By including the String representation of the DPID in each logged message, the message can be tracked back to the Diagnostic Provider related to the component. A method is provided to associate Diagnostic Provider IDs with Loggers (from the *java.util.logging* logging API).



The previous diagram shows how the use of DPIDs in log entries enables callbacks to the component that originally created the log entry.

1. Shows the component logging with a DPID included in the log entry.
2. The administrator examines the log entry through the Runtime Messages Utility and notices that the entry has a link to a Diagnostic Provider.
3. The administrator uses the link to gain access to the relevant MBean in the Diagnostic Provider Utility .
4. The Diagnostic Provider Utility contacts the Diagnostic Provider MBean to ask for more information.
5. The request for more information is sent back to the source of the original log entry.
6. The response from the Diagnostic Provider is provided in the administration console.

Diagnostic Provider configuration dumps, state dumps, and self tests

The Diagnostic Provider (DP) infrastructure allows for a software component or stack product in the WebSphere Application Server space to expose key information about its configuration, current state, and current ability to perform operations.

The methods that expose this information might be driven as a result of a message put out by the component (by a logger which automatically includes the Diagnostic Provider ID in each message), or might be driven as a result of an overall system health-check when an administrator or automated tool is monitoring the system.

Configuration dumps

A *Configuration dump* is an operation you can perform on a Diagnostic Provider to list the startup or current values of the configuration attributes for the DP. The name for each data item in this dump should reflect its disposition. That is, each item should be called *startup-xxx* or *current-xxx* to show whether this is a startup or current value. The collection of attributes returned from this operation can be thought of as the **payload** of the configuration dump. More information about payloads can be found in “Diagnostic Provider method implementation” on page 298.

You can find several ways to filter the output of a configuration dump in “Diagnostic Provider registered attributes and registered tests” on page 293.

State dumps

A *State dump* is similar to a configuration dump, but it differs in two key areas. First, a state dump displays current information about the operation of a component. An example is a connection pool. A configuration dump can show *DataSource name*, the *minConnections* (configured or current), the *maxConnections*, the *DataBase name*, and so on. A state dump is more likely to show the current connections in use, the high concurrent use count, the number of times the pool has been expanded, the average time between requesting a connection and returning it, and so forth.

State dumps can be impacted by the values in the State Collection Specification. This is a dynamic specification that controls additional data collection that the component can do at runtime. If additional data is being collected, then a State dump might display more information. The same filters and payload information that apply to Configuration dumps (see “Diagnostic Provider registered attributes and registered tests”) apply to State dumps.

Self Diagnostic tests

Self diagnostic tests are non-invasive operations that a Diagnostic Provider exposes. *Non-invasive* means that if they modify anything for the test, the conclusion of the test reverses the modification. These tests give an administrator the option to test simple functions of a component to see if it is able to perform them.

The filters for a self diagnostic test apply to the test itself, not to the output of the test. A typical use of Self Diagnostic tests could be for a pool manager of some sort to pull an object out of the pool and return it to the pool to verify that this operation can still be performed, and with acceptable performance.

Diagnostic Provider registered attributes and registered tests

Each Diagnostic Provider (DP) provides a list of state dump attributes, configuration dump attributes, self tests, and self test attributes. The tests are operations that the DP can perform. The attributes are pieces of information that are available for collection from a Configuration dump, a State dump, or a specific Self Diagnostic test.

Each attribute can be seen as a piece of information with a label on it. Each attribute is also considered to be either *registered* or *not registered*. A registered attribute is one that should be available from one release of WebSphere Application Server to the next. A nonregistered attribute might not be available in its current form in future releases of the product (no commitment has been made).

When performing a Configuration dump, a State dump, or a Self Diagnostic test, an administrator or automatic tooling can request *only registered* values, or *all* values, depending on the needs of the administrator or tool. Note that the option of filtering results is only available through the Diagnostic Provider's Java Management Extension (JMX) MBean interface, which you can access programmatically or through the wsadmin tool.

The DiagnosticProviderRegistration XML file

The DiagnosticProviderRegistration Extensible Markup Language (XML) file is used in conjunction with the method signatures to filter the results of calling the various methods. This XML file defines the configuration information, state information, and self diagnostic tests exposed by the component. In the configuration and state information, the key working unit is referred to as the attribute. Specification of an attribute is as follows:

```
<attribute>
        <id><Regular Expression representing the attribute name></id>
        <descriptionKey><MsgKey into a ResourceBundle for localization of the label></descriptionKey>
        <registered>true</registered>
   </attribute>
```

The parts are as follows:

ID: The attribute's name. This name can be expressed with wildcard characters conforming to regular expression syntax. The registered attribute ID is used in the following places:

- Within Diagnostic Provider configuration dump and state dump methods to determine which attributes to return.
- In the administration console to match description keys to attributes returned from a request to a Diagnostic Provider for a configuration dump, state dump, or self diagnostic operation.

As an example, if a configuration dump returns an attribute with ID **cachedServlet-MyServlet-servletPath** to the administration console, the administration console could use the *descriptionKey* corresponding to the attribute registered as **<id>cachedServlet-.*-servletPath</id>** when selecting what description text to put next to this attribute's name and return values.

descriptionKey:
This is a key into a *resourceBundle* for localization.

registered:
This is a boolean qualifying whether this attribute will be available from one release of the software to the next. If registered is **true**, then this attribute should be available in the next release. If registered is **false**, then there is no guarantee that this attribute will continue to exist. Automation should use some caution when handling non-registered attributes.

Specification of a selfDiagnosticTest is as follows:

```
<test>
       <id><Regular Expression for the name of the test></id>
       <descriptionKey><MsgKey into a ResourceBundle for localization of the label></descriptionKey>
       <attribute><One or more attributes which will be output from this test></attribute>
</test>
```

The parts are as follows:

ID: Similar to the ID for the attribute, but in this case, describing the test to be performed instead of the attribute to be returned.

descriptionKey:
This is a key into a resourceBundle for localization.

Method interfaces

```
public DiagnosticEvent [] configDump(String aAttributeId, boolean aRegisteredOnly);
public DiagnosticEvent [] stateDump(String aAttributeId, boolean aRegisteredOnly);
```

These methods invoke the configuration or state dump on the component, and specify a regular expression filter for the attributes to return as well as filtering the output to include all matching attributes, or only those attributes which are registered. This enables the administrator or automated software driving the method to specify a subset of the overall fields (especially important if many attributes are exposed or if the State Collection Specification increases the amount of data available). The following helper methods are available to assist with filtering the output.

To take a list of Attributes that are available to return, and filter them:

```
    public static AttributeInfo [] queryMatchingDPInfoAttributes(String aAttributeId,
        AttributeInfo [] inAttrs, String [] namesToCheck, boolean aRegisteredOnly) {
```

To take a single Attribute that is available to return, and filter it:

```
    public static AttributeInfo queryMatchingDPInfoAttributes(String aAttributeId,
        AttributeInfo [] inAttrs, String nameToCheck, boolean aRegisteredOnly) {
```

To go through a populated set of Attribute Information and remove unneeded parts:

```
    public static void filterEventPayload(String aAttributeId, HashMap payLoad) {
```

For details on these messages, please review the API documentation for the **DiagnosticProviderHelper** class. The basic concept is that, once the component knows what attributes are able to be returned, the helper method will determine which of them should be returned based on the regular expression logic and registration boolean.

The selfDiagnostic Method interface here is similar to that of Configdump and Statedump:

```
public DiagnosticEvent[] selfDiagnostic(String aTestId, boolean aRegisteredOnly)
```

The difference is that the first parameter is a regular expression filter for which test to run.

Diagnostic Provider names

In addition to the Diagnostic Provider ID (DPID), each component that implements the Diagnostic Provider interface must have a Diagnostic Provider Name. While the DPID must be unique within the entire WebSphere Application Server domain, the Diagnostic Provider Name need only be unique within the Java Virtual Machine (JVM).

Unlike the Diagnostic Provider ID, which tends to be long and not human-friendly, the Diagnostic Provider Name should be shorter and easier to read. In addition, by convention it should end in *DP*. The Diagnostic Service MBean (see "The simpler interfaces provided by the Diagnostic Service MBean") can drive methods on a Diagnostic Provider using its name.

The simpler interfaces provided by the Diagnostic Service MBean

All services for a Diagnostic Provider (DP) are also available through a Java Management Extensions (JMX) interface known as the *Diagnostic Service* interface. The Diagnostic Service interface enables administrators to drive methods against DPs using the Diagnostic Provider Name or Diagnostic Provider ID.

When formatted output is requested of the Diagnostic Service, it is localized to the client Locale. This makes the Diagnostic Service MBean ideal for clients using an interface where consuming complex Java objects, such as those returned from the Diagnostic Provider MBeans, is not feasible. An example of such an interface is the wsadmin tool.

The Diagnostic Service interface provides four signatures for each of the key methods available on the Diagnostic Providers (*configDump*, *stateDump*, and *selfDiagnostic*) objects. Because these method signatures look so similar, this example shows it all through the configDump methods. The four Diagnostic Service methods that map to configDump on a Diagnostic Provider are:

```
public DiagnosticEvent [] configDump(String aDPName, String aAttributeIdSpec, boolean aRegisteredOnly)
public DiagnosticEvent [] configDumpById(String aDPid, String aAttributeIdSpec, boolean aRegisteredOnly)
public String [] configDumpFormatted(String aDPName, String aAttributeIdSpec,
       boolean aRegisteredOnly, Locale aLocale)
public String [] configDumpFormattedById(String aDPid, String aAttributeIdSpec,
       boolean aRegisteredOnly, Locale aLocale) {
```

The first two return exactly what the Diagnostic Provider does. The second two methods act as a pass-through to the actual Diagnostic Provider, but they take the array of Diagnostic Events that the Diagnostic Provider returns, and convert it into a more easily consumable *String* array. In addition, these methods handle localizing the output to the appropriate locale. It is important to note that the same method can be driven using the Diagnostic Provider ID or the Diagnostic Provider Name.

Creating a Diagnostic Provider

Use Diagnostic Providers to query the startup configuration, current configuration, and current state of a diagnostic domain. Diagnostic Providers also provide access to any self diagnostic tests that are available from a diagnostic domain.

Before you begin

To complete this task you must have programming knowledge of your system and the proper authorities to perform the following steps.

About this task

The steps that follow outline a general process for creating Diagnostic Providers (DP).

Procedure

1. Determine your diagnostic domain. Look for *configuration* MBeans that control a similar domain in the same component. Extending an existing configuration MBean with a DP interface avoids proliferation of new MBeans and has the benefit that mapping from a diagnostic MBean to a configuration MBean requires no additional information.
2. Determine what configuration attributes you want to expose. Include information that is used to configure your component from the configuration MBeans.
3. Determine what state attributes you want to expose. Anything you might want to know about the state of your component for troubleshooting can go here.
4. Determine what self diagnostic tests you will expose.
5. Determine what test attributes you will return for each self diagnostic.
6. Create your DP registration Extensible Markup Language (XML) file.
7. Create your DP implementation.
   a. To see an example, refer to “Implementing a Diagnostic Provider” on page 298 and keep in mind that most things that a Diagnostic Provider should do are already done for you in the *DiagnosticProviderHelper* class.
   b. To ensure that you do not collect unwanted data, add hooks in your component code where you need to collect state data using the *DiagnosticConfig* object.
   c. Add hooks in your component code where you need to store or be able to access configuration data.
8. Add code to register your DP implementation. Typically, the best place to do this is where your component is initialized.
9. Add Diagnostic Provider IDs (DPID) to your logged messages. Registering a DPID with a logger makes that information available in any messages logged with this logger. This enables fast paths in the DP utility to function on this particular Diagnostic Provider.
   a. Register your DPID with your loggers (for any of your loggers that you only want to associate a single DPID with).
   b. When you use multiple DPIDs with the same Logger, you can (instead of registering a single DPID with a Logger) add DPIDs to individual logging calls in the **parm[0]** position. Do not put **{0}** in the corresponding localized messages. It is bad practice to print the DPID in your messages as this would be inconsistent with messages from loggers with statically assigned DPIDs.

Diagnostic Provider Extensible Markup Language

Some conventions to follow for Diagnostic Provider (DP) Extensible Markup Language (XML) declarations.

These guidelines are to help keep your use of Diagnostic Providers (DP) consistent.

- Include the Document type definition (DTD) for your Diagnostic Provider at the top of every DP declaration Extensible Markup Language (XML) file.
- Start all names and name segments with lower case. Use `camel case` for attribute names. That is, capitalize every initial letter in the name, except the first. For example, *traceCollectionSpec*.
- Indicate hierarchy with dashes. Dashes work better than dots because attribute names are regular expressions. For example, `traceService-traceCollectionSpec`.

- Indicate string dynamic parts to attribute names using an asterisk (*). For example,
  `vhosts-.*-webgroups-.*-webapps-.*-listeners-filterInvocationListeners`

  which would match vhosts-*someHost*-webgroups-*someGroup*-webapps-*someApp*-listeners-filterInvocationListeners
- Indicate numeric dynamic parts to attribute names using *[0-9]**. For example,
  `vhosts-index-[0-9]*`

  which would match webcontainer-vhosts-index-123
- If you have a general purpose self diagnostic test that can be run without significant performance cost, name it `general`.

Some tips for configDump implementation

- configDump should contain information used to define the component's environment. Some examples are:
  - configuration data set by Java Management Extensions (JMX)
  - configuration from system properties, xml files, and property files
  - configuration information hard-wired and unchanging in code (such as, if a resource adapter implements interface X, or has some static final field Y, then those could indicate aspects of configuration and be included in the configDump).
- configDump should not contain dynamically registered attributes, such as:
  - a list of registered loggers (this belongs in stateDump)
  - a list of servlets in an application (this belongs in stateDump).
- configDump should be separated into two sections -- *startup* and *current*.
  - All configDump attributes must start with either *startup-* or *current-*.
  - The *startup* section details the component's environment at startup time. Startup configDump attributes start with *startup-* .
  - The *current* section details the component's environment at the moment the configDump is requested. Current configDump attributes start with *current-*.

Best practices for configDump

- Group related attributes using an attribute hierarchy (such as, for two attributes about the traceLog: startup-traceLog-rolloverSize=20, startup-traceLog-maxNumberOfBackupFiles=1)
- For information in the current attribute list that refers to the same thing as a startup attribute, the names of both current and startup attributes should match.
- If an attribute has no use following startup, only show it in the startup section (for example, a configuration attribute that contains a file name from which startup data is read).

Choosing a Diagnostic Provider name

To ensure consistency when choosing Diagnostic Provider names to use with your components, you should consider the guidelines that follow.

Diagnostic Provider name guidelines:

- Names must be unique within a Java Virtual Machine (JVM). One Diagnostic Provider name goes uniquely with one Diagnostic Provider ID within a server.
- If necessary, names can contain a dynamic element to help with uniqueness. Of course, the dynamic element should have meaning to the administrator.
- Although not a hard limit, the static part of names should be 16 characters or less.
- The static part of names must follow the class name convention. Start with a capital letter, no spaces, and capitalize each word in the name.

- The static part of names must end with **DP**.
- Valid names contain a static part only, or a static part followed by a dash (-), followed by a dynamic part. Some valid examples:
  – ConnMgrDP-instance_specific_stuff
  – WebContainerDP
  – AdvisorDP
  – NodeAgentDP

Implementing a Diagnostic Provider

To use a Diagnostic Provider you must configure an MBean with the methods and attributes required to handle the data from the application server and client applications.

Before you begin

This task presumes that you have a programming knowledge of the creation of MBeans. For more information about the interaction of MBeans with WebSphere Application Server, refer to topic, Creating and registering standard, dynamic, and open custom MBeans in the *Administering applications and their environment* PDF book.

About this task

The steps that follow outline a general process for implementing a Diagnostic Provider (DP).

Procedure

1. Modify the MBean descriptor Extensible Markup Language (XML). To implement a Diagnostic Provider, you must have an MBean, and the MBean should include this statement in its descriptor XML as a direct child of the MBean element:

   ```
   <parentType type="DiagnosticProvider"/>
   ```

   This defines the operations, attributes, and aggregators necessary for an MBean to be a Diagnostic Provider. If you do not need to have this DP exist in z/OS Controllers, then this XML inclusion handles all z/OS specifics for your MBean.
2. Modify the MBean Implementation. Your MBean should already have a class which instantiates it and registers it with the Java Management Extensions (JMX) server.

   The first difference here is that you must define a property in the *Properties* class that is passed to the registration (and becomes part of the *ObjectName*). The property is **diagnosticProvider=true** and it can be added with a line of code such as:

   ```
   MyProps.setProperty(DiagnosticProvider.DIAGNOSTIC_PROVIDER_KEY, DiagnosticProvider.DIAGNOSTIC_PROVIDER_VALUE) ;
   ```

   The second difference is that this class should register this Diagnostic Provider with the Diagnostic Service. A helper method is available to do this:

   ```
   DiagnosticProviderHelper.registerMBeanWithDiagnosticService(DiagnosticProviderPName, DiagnosticProviderId) ;
   ```

   Obviously this must be done after the registration when the *ObjectName* can be retrieved into the *DiagnosticProviderId* string.
3. Implement the Diagnostic Provider methods.

Diagnostic Provider method implementation

To create a Diagnostic Provider (DP) you must have an MBean that includes the required methods in its deployment Extensible Markup Language (XML) file. These methods define the operations, attributes, and aggregators necessary for an MBean to be a Diagnostic Provider.

Adding these methods can be accomplished by adding the *parentType* directive to your existing XML file (see “Implementing a Diagnostic Provider”), or by including the operations directly into your deployment

XML file. The definitions needed are included in “Diagnostic Provider registered attributes and registered tests” on page 293. The next step is for the MBean to actually implement these methods. The methods to implement include:

- “getRegisteredDiagnostics”
- “getDiagnosticProviderName”
- “getDiagnosticProviderID”
- “configDump”
- “stateDump” on page 300
- “selfDiagnostic” on page 300
- “localize” on page 301

getRegisteredDiagnostics

This method exposes the registration information for this Diagnostic Provider. It is commonly used by the DP Utility in the administration console to gather information about Diagnostic Providers that are to be displayed in the console. This method returns a **DiagnosticProviderInfo** object that is usually attained by passing the appropriate XML to a **DiagnosticProviderHelper** helper class. Here is an example:

```
public DiagnosticProviderInfo getRegisteredDiagnostics() {
  InputStream regIS= Thread.currentThread().getContextClassLoader().getResourceAsStream(
   "com/ibm/ws/xxx/SampleDP2DiagnosticProvider.xml");
  dpInfo = DiagnosticProviderHelper.loadRegistry(regIS, sDPName) ;

         if (dpInfo == null) {
            sSampleDP2MBeanLogger.logp(Level.WARNING, sThisClass, "getRegisteredDiagnostics",
             "RasDiag.DPInfo.NoGotz") ;
  }
    return dpInfo ;
 }
```

Notice that the XML is packaged and available in the *classpath* of the current *classloader*. The “Registration XML” on page 302 contains crucial information that the Diagnostic Provider uses to “Populate the payload” on page 301 and “localize” on page 301 results.

getDiagnosticProviderName

This is usually a pretty simple return of a constant as the following example shows

```
public String getDiagnosticProviderName() {
  return sDPName;
 }
```

getDiagnosticProviderID

This is usually a pretty simple return of a Java Management Extensions (JMX) object ID that MBeans can pull out of the base class method. For example:

```
public String getDiagnosticProviderId() {
  return getObjectName().toString() ;
 }
```

configDump

The *configDump* method enables the Diagnostic Provider to expose the configuration data that was in place when this Diagnostic Provider started (or the current values of them). The **DiagnosticEvent** objects that this method returns include a “Payload” on page 301 that contains the core data. The following is an excerpt from a configDump method:

```
public DiagnosticEvent [] configDump(String aAttributeIdSpec, boolean aRegisteredOnly) {
    HashMap cdHash = new HashMap(64) ;

  // "Populate the payload" on page 301
```

```
    DiagnosticEvent [] diagnosticEvent = new DiagnosticEvent[1] ;
    diagnosticEvent[0] = DiagnosticEventFactory.createConfigDump(getObjectName().toString(),
     "ThisClassName", "configDump", cdHash) ;

    return diagnosticEvent ;
 }
```

This returns an array of **DiagnosticEvent** objects. Normally, *configDump* and *stateDump* return only one object. However, the method accepts an array because on z/OS systems a server can have multiple servants, and aggregation of the output from the servants is stored in the array.

stateDump

The *stateDump* method enables the Diagnostic Provider to expose the current state data, or data about the current operating conditions of the Diagnostic Provider. The data made available can be anything likely to assist a customer, an IBM support person, or automated tooling in analyzing the health of the component and problem determination if there is an issue. The amount of data available is impacted by the State Collection Specification in effect at the time. If the current State Collection Specification involves the collection of additional data by the Diagnostic Provider, then this additional data can be exposed in the stateDump. The **DiagnosticEvent** objects that this method returns include a “Payload” on page 301 that contains the core data. The following is an excerpt from a stateDump method:

```
public DiagnosticEvent [] stateDump(String aAttributeIdSpec, boolean aRegisteredOnly) {
    HashMap sdHash = new HashMap(64) ;

  // "Populate the payload" on page 301

    DiagnosticEvent [] diagnosticEvent = new DiagnosticEvent[1] ;
    diagnosticEvent[0] = DiagnosticEventFactory.createStateDump(getObjectName().toString(),
     "ThisClassName", "stateDump", sdHash) ;

    return diagnosticEvent ;
 }
```

This returns an array of **DiagnosticEvent** objects. Normally, *configDump* and *stateDump* return only one object.

The method accepts an array because a z/OS server can have multiple servants, and aggregation of the output from the servants is stored in the array.

selfDiagnostic

The *selfDiagnostic* method enables the Diagnostic Provider to perform certain predefined activities to test key functionalities of your system. These tests should not have a lasting effect on the system. For example, if the test is to create a TCP/IP connection to a remote host, the test should also break that connection before returning its results so that the state of the component is unchanged by the test. The information returned by the test is determined by the attributes included in the test section of the XML file. The following is an excerpt from a selfDiagnostic method:

```
public DiagnosticEvent [] selfDiagnostic(String aAttributeIdSpec, boolean aRegisteredOnly) {
  TestInfo [] testInfo = dpInfo.selfDiagnosticInfo.testInfo ; // Retrieve the test registry information
  Pattern testChecker = Pattern.compile(aAttributeIdSpec) ; // Compile test regexp parm for faster checking
  ArrayList deList = new ArrayList(8) ;    // Allocate expandable list of DiagnosticEvents
  for (int i = 0; i < testInfo.length; i++) {
   if (testChecker.matcher(testInfo[i].id).matches()) {
      HashMap deHash = new HashMap(32) ;

    // "Populate the payload" on page 301

      deList.add(DiagnosticEventFactory.createDiagnosticEvent(getObjectName().toString(),
       DiagnosticEvent.EVENT_TYPE_SELF_DIAGNOSTIC, DiagnosticEvent.LEVEL_INFO,
```

```
      "ThisClassName", "selfDiagnostic", dpInfo.resourceBundleName,
    "RasDiag.SDP2.createDE3",   // MsgKey for localization
          // Parms to incorporate in msg
    new Object [] { "OneParm", "TwoParm", "RedParm", "BlueParm"}, deHash)) ;
   }
  }

    DiagnosticEvent [] diagnosticEvent = new DiagnosticEvent[deList.size()] ;
  diagnosticEvent = (DiagnosticEvent [])deList.toArray(diagnosticEvent) ;

    return diagnosticEvent ;
 }
```

This returns an array of **DiagnosticEvent** objects. In this example, one **DiagnosticEvent** was created from each test that matched the parameter regular expression. The Diagnostic Provider is not required to produce only one per test. The generation of “Payload” is similar to that of *configDump* and *stateDump*.

Aggregation on multiple z/OS servants for an individual server concatenates the arrays from each servant.

localize

The **DiagnosticEvents** that methods return contain payload **HashMaps** that contain **MessageKeys** and **ResourceBundles**. The final consumer of these events is often not on the server, and thus may not have the appropriate *classpath* to resolve this. For this purpose, a callback to the Diagnostic Provider to localize the variables is done. A helper method, however, makes it a simple method to write, as this example demonstrates:

```
public String [] localize(String [] aKeys, Locale aLocale) {
  return DiagnosticProviderHelper.localize(dpInfo.resourceBundleName, aKeys, aLocale) ;
 }
```

Note that the **dpInfo** (DiagnosticProviderInfo) object is needed as this object includes a reference to the **ResourceBundle**.

Payload

A recurring theme in these methods is the ability to include a payload in return objects. This is a set of *name=value* pairs that include the information being exposed by the method. Diagnostic Events returned from a configDump, stateDump, or selfDiagnostic test are relatively complex Java objects. The majority of the information that is returned is contained in the **DiagnosticData** portion of the **DiagnosticEvent** object. Each attribute returned by the Diagnostic Provider is stored in an entry in a **HashMap**. There can be cascading HashMaps within a single DiagnosticEvent object (if breaking the data down into subGroups makes sense). Each HashMap entry contains either a reference to a child HashMap, or a **DiagnosticTypedValue** (which contains the value, the type of data, and a **MsgKey** for localization of the label or /name). The values to be returned should be filtered with:

- The type of method (that is, configDump, stateDump, or selfDiagnostic)
- The **AttributeIdSpec** sent in to filter the values
- The current State Collection Specification (which can impact the amount of data available).

Populate the payload

The API documentation for *DiagnosticProviderHelper.queryMatchingDPInfoAttributes* explains how to do the filtering before retrieving the data. In some cases, it is easier and helps performance for a Diagnostic Provider to retrieve all data into the Payload and then filter the HashMap after the fact. The post-population filtering can be done with the method *DiagnosticProviderHelper.filterEventPayload*. For information on use of the JavaBeans type approach, see the API documentation for the *AttributeBeanInfo.populateMap* method.

Registration XML

Registration XML enables much of the information needed by the Diagnostic Provider to be externalized. It also provides a means of commonizing localization and consumption of the tests (thus aiding automation). An excerpt of this XML from a sample Diagnostic Provider follows:

```
<!DOCTYPE diagnosticProvider PUBLIC "RasDiag" "/DiagnosticProvider.dtd">

<diagnosticProvider>
    <resourceBundleName> com.ibm.ws.rasdiag.resources.RasDiagSample</resourceBundleName>
    <state>
 <attribute>
  <id>Leg-Foot</id>
  <descriptionKey>SampleDiagnostic.LegFoot.descriptionKey</descriptionKey>
  <registered>true</registered>
 </attribute>
 <attribute>
  <id>Leg-Ankle</id>
  <descriptionKey>SampleDiagnostic.LegAnkle.descriptionKey</descriptionKey>
  <registered>true</registered>
        </attribute>
    </state>
    <config>
        <attribute>
  <id>Arm-Hand-Size</id>
  <descriptionKey>SampleDiagnostic.HandSize.descriptionKey</descriptionKey>
  <registered>true</registered>
        </attribute>
        <attribute>
  <id>Leg-Foot-Size</id>
  <descriptionKey>SampleDiagnostic.FootSize.descriptionKey</descriptionKey>
  <registered>true</registered>
        </attribute>
    </config>
    <selfDiagnostic>
        <test>
  <id>Kick</id>
  <descriptionKey>SampleDiagnostic.Kick.descriptionKey</descriptionKey>
  <attribute>
   <id>Kick-Pain</id>
   <descriptionKey>SampleDiagnostic.KickPain.descriptionKey</descriptionKey>
  </attribute>
  <attribute>
   <id>Kick-Length</id>
   <descriptionKey>SampleDiagnostic.KickLength.descriptionKey</descriptionKey>
  </attribute>
        </test>
 <test>
  <id>Throw</id>
  <descriptionKey>SampleDiagnostic.Throw.descriptionKey</descriptionKey>
         <attribute>
   <id>Throw-Pain</id>
   <descriptionKey>SampleDiagnostic.ThrowPain.descriptionKey</descriptionKey>
   <registered>true</registered>
  </attribute>
  <attribute>
   <id>Throw-Length</id>
   <descriptionKey>SampleDiagnostic.ThrowLength.descriptionKey</descriptionKey>
   <registered>true</registered>
  </attribute>
        </test>
    </selfDiagnostic>
</diagnosticProvider>
```

For understanding the storage of this information into a **DiagnosticProviderInfo** object, see the API documentation for *DiagnosticProviderInfo*. For conceptual information about the purpose of the registration

XML, see “Diagnostic Provider registered attributes and registered tests” on page 293.

Diagnostic Provider XML example:

Here is an example of the Diagnostic Provider Extensible Markup Language (XML).

```
version="6.0"
platform="common"
aggregationHandlerClass="com.ibm.ws.management.component.DiagnosticProviderAggregator"
description="DiagnosticProvider portion of Mbean for inclusion into MBeans implementing this interface">
  <attribute
    description="DiagnosticProviderName (not dependent on runtime, but subset of ObjectName"
    getMethod="getDiagnosticProviderName" name="diagnosticProviderName"
    type="java.lang.String" proxyInvokeType="unicall" proxySetterInvokeType="multicall"/>
  <operation
    description="Get the DiagnosticProvider ID"
    impact="INFO" name="getDiagnosticProviderId" role="operation"
    targetObjectType="objectReference" type="java.lang.String" proxyInvokeType="unicall">
    <signature/>
  </operation>
  <operation
    description="Return the registry information based on type (config/state/selfDiag)."
    impact="INFO" name="getRegisteredDiagnostics" role="operation"
    targetObjectType="objectReference"
    type="com.ibm.wsspi.rasdiag.diagnosticProviderRegistration.DiagnosticProviderInfo"
    proxyInvokeType="unicall">
    <signature/>
  </operation>
  <operation
    description="Dump the configuration information associated with managed resource."
    impact="INFO" name="configDump" role="operation"
    targetObjectType="objectReference" type="[Lcom.ibm.wsspi.rasdiag.DiagnosticEvent;"
    proxyInvokeType="multicall">
    <signature>
      <parameter description="Attribute ID to use"
        name="attributeId" type="java.lang.String"/>
      <parameter description="Report on just registered info, or all info"
        name="registeredOnly" type="boolean"/>
    </signature>
  </operation>
  <operation
    description="Dump state information for the managed resource."
    impact="INFO" name="stateDump" role="operation"
    targetObjectType="objectReference" type="[Lcom.ibm.wsspi.rasdiag.DiagnosticEvent;"
        proxyInvokeType="multicall">
    <signature>
      <parameter description="Attribute ID to use"
        name="attributeId" type="java.lang.String"/>
      <parameter description="Report on just registered info, or all info"
        name="registeredOnly" type="boolean"/>
    </signature>
  </operation>
  <operation
    description="Perform diagnostics on the managed resource driven by current diagnostic mode setting."
    impact="ACTION" name="selfDiagnostic" role="operation"
    targetObjectType="objectReference" type="[Lcom.ibm.wsspi.rasdiag.DiagnosticEvent;"
        proxyInvokeType="multicall">
    <signature>
      <parameter description="Test ID to use"
        name="testId" type="java.lang.String"/>
      <parameter description="Report on just registered info, or all info"
        name="registeredOnly" type="boolean"/>
    </signature>
  </operation>
  <operation
    description="localize messages for console display"
    impact="INFO" name="localize" role="operation"
```

```
    targetObjectType="objectReference" type="[Ljava.lang.String;"
        proxyInvokeType="unicall">
    <signature>
      <parameter description="Message Keys" name="msgKeys" type="[Ljava.lang.String;"/>
      <parameter description="Locale to use for output" name="locale" type="java.util.Locale"/>
    </signature>
  </operation>
```

Creating a Diagnostic Provider registration XML file

The Diagnostic Provider registration XML is used to provide information about the exposed configuration, state, and self diagnostic attributes and tests to the Diagnostic Provider utility. It is also used to populate objects needed later in the process, to assist in filtering, and to assist in localization.

Before you begin

Programming knowledge of your system and the proper authorities to perform the following steps.

About this task

The steps that follow outline a general process for creating a Diagnostic Provider (DP) registration Extensible Markup Language (XML) file.

Procedure

1. Start with the DP document type definition (DTD). If you are using the helper methods (see the step called *Create your DP implementation* in “Creating a Diagnostic Provider” on page 295), you can use this DOCTYPE line to pick up the common DTD:

   ```
   <!DOCTYPE diagnosticProvider PUBLIC "RasDiag" "/DiagnosticProvider.dtd">
   ```

   If you are extending an existing MBean with an existing XML configuration, you might need either to add the DP XML to an existing DTD, or omit the DP XML entirely. If you omit the DP XML, you will not be able to validate that your XML file is well formed.
2. Follow the conventions described in “Diagnostic Provider Extensible Markup Language” on page 296 to help keep your XML consistent with other components. You can find an example of a small DP registration XML file in “Diagnostic Provider method implementation” on page 298.

Associating a Diagnostic Provider ID with a logger

If you are using a Diagnostic Provider to manage alerts and messages, you need to associate the Diagnostic Provider ID with a logger. This can be done dynamically or through a static assignment.

About this task

Components whose diagnostics are managed through a Diagnostic Provider MBean should include the Diagnostic Provider ID (DPID) in all logged messages. In some cases a single logger always logs with the same DPID. In those cases, it is appropriate to statically associate the DPID with the logger. In other cases, a logger might log on behalf of various diagnostic domains. For example, although every data source has a separate Diagnostic Provider MBean, they all share the same logger. In those cases, the DPID can be dynamically supplied on each logging call.

Static Assignment

About this task

The method below statically assigns a DPID to a logger.

Procedure

Associate a DPID with a logger:

```
Logger logger = Logger.getLogger("com.ibm.ws.MyClass");
DiagnosticProviderHelper.addDiagnosticProviderIDtoLogger(logger, dpid);
```

Dynamic Assignment

About this task

DPIDs can be associated with a single log request by including them as the first message parameter, prefixed with **DPID:.** To associate a DPID with a single log request using a logger:

```
Object[] parms = new Object[] { "DPID:" + dpid };
logger.logp(classname, methodname, "MSG0001", parms);
```

Note that in the dynamic case, the DPID does not need to actually show up in the formatted message. The two examples below illustrate:

```
(in resource bundle)
// by not including {0} first parm is not printed in the message.
MSG0001=This message does not include the DPID.

// note - it is not recommended to print the DPID in your message.
MSG0002=This message includes the DPID...it's value is {0}.
```

It is recommended that messages not include the DPID in the formatted message. As shown above, this is done by not including {0} in the message value in the resource bundle.

Using Diagnostic Providers from wsadmin scripts

In addition to enabling Diagnostic Providers (DP) from the administration console, you can also use them through scripts from the Wsadmin tool.

About this task

You might want to enable, disable, or configure Diagnostic Providers from the administrative console, but in some cases it might be more efficient or useful to do so with scripts using the wsadmin tool.

Read the wsadmin tool information about using the tool with scripts.

Procedure

1. List the MBeans that implement the Diagnostic Provider (DP) interface. Enter

   ```
   $AdminControl queryNames diagnosticProvider=true,*
   ```

   And you will see an output that displays all of the Diagnostic Providers in a format like this:

   ```
   "WebSphere:name=Default Datasource,process=server1,platform=dynamicproxy,node=
     camelhair,JDBCProvider=Derby JDBC Provider,
   diagnosticProvider=true,j2eeType=JDBCDataSource,J2EEServer=server1,Server=server1,
     version=6.1.0.0,type=DataSource,
   mbeanIdentifier=cells/camelhairCell/nodes/camelhair/servers/server1/resources.xml#
     DataSource_1131113688564,
   JDBCResource=Derby JDBC Provider,cell=camelhairCell"
   "WebSphere:name=DefaultEJBTimerDataSource,process=server1,platform=dynamicproxy,
     node=camelhair,
   JDBCProvider=Derby JDBC Provider (XA),diagnosticProvider=true,j2eeType=
     JDBCDataSource,J2EEServer=server1,Server=server1,version=6.1.0.0,type=DataSource,
     mbeanIdentifier=cells/camelhairCell/nodes/camelhair/servers/server1/
     resources.xml#DataSource_1000001,
   JDBCResource=Derby JDBC Provider (XA),cell=camelhairCell"
   ```

```
WebSphere:name=WebcontainerDiagnosticProvider,process=server1,platform=
  dynamicproxy,node=camelhair,diagnosticProvider=true,
version=6.1.0.0,type=WebcontainerEventProvider,mbeanIdentifier=null,
  cell=camelhairCell
```

2. Capture the ObjectName of your Diagnostic Provider in a variable. This enables you to reference your Diagnostic Provider more easily, especially in a script. For example, instead of typing all of those lines, if you want to work with the WebContainer Diagnostic Provider, for example, you can do the following:
   - set DP [lindex [$AdminControl queryNames name=WebcontainerDiagnosticProvider,diagnosticProvider=true,*] 0]

     This ObjectName stored in the DP variable can be used on the methods, or you can use the Diagnostic Provider name as text or a variable.
   - Now that you have the ObjectName in a variable, you can get the Diagnostic Provider name in a variable with the command:

     ```
     set DPNm [$AdminControl invoke $DS getDiagnosticProviderNameById $DP]
     ```

     This provides the result:

     ```
     WebContainerDP
     ```

     Now the DiagnosticProvider (WebContainer) is addressable by its objectname in variable DP, or by its DiagnosticProvider name in variable DPNm. If you would prefer, you can hard-code the DPName *WebContainerDP* as it is short enough.
3. Save the ObjectName of the DiagnosticService MBean to a variable. For wsadmin, WebSphere Application Server provides this MBean so that the output of the Diagnostic Provider is more easily consumable. Enter

   ```
   set DS [lindex [$AdminControl queryNames name=DiagnosticService,*] 0]
   ```
4. Run a configDump. You can run a configDump and capture all attributes with the command:

   ```
   $AdminControl invoke $DS configDumpFormattedById [list $DP .* true null]
   ```

   This lists the values that the Diagnostic Provider used at start up (and possible current values). .

Table 98. An excerpt of the configDump output. The following table lists the values that the Diagnostic Provider used at start up and possible current values.

Item Concatenated Name	Value
customProperties =	Null
defaultVirtualHostName =	default_host
jvmProps =	Null
localeProps =	Null
servletCachingEnabled =	false
aliases =	*:9080;*:80;*:9443;

5. Filter the output of your configDump. You can use configDumpFormatted (leaving off the ById) and switch **$DP** for **$DPNm** or the string **WebContainerDP**. This example uses *$DPNm* on this slightly modified version whereby it only picks up attributes dealing with automation:

   ```
   $AdminControl invoke $DS configDumpFormatted [list $DPNm .*auto.* true null]
   ```

   This results in just those attributes that contain **auto** in them. Full (but strict) regular expression syntax is allowed.

Table 99. Results. The following table lists the concatenated names and values.

Item Concatenated Name	Value
autoLoadFiltersEnabled =	false
autoRequestEncoding =	false
autoResponseEncoding =	false
autoLoadFiltersEnabled =	false

Table 99. Results (continued). The following table lists the concatenated names and values.

Item Concatenated Name	Value
autoRequestEncoding =	false
autoResponseEncoding =	false

The syntax is the same for stateDumps and selfDiagnostics

Viewing the run time configuration of a component using Diagnostic Providers

You can use the administrative console to navigate to configuration data that can be used to check the health of a server runtime component.

Before you begin

You must have sufficient authority to run the action.

About this task

Runtime components that have associated diagnostic providers can include their Diagnostic Provider ID (DPID) in their log entries. If you know the DPID, you can enter it directly in the quick link text box. Otherwise, navigate to the desired process by using the tree view displayed at the bottom of the panel, as shown in the steps below.

Procedure

1. Start the administration console.
2. From the task bar on the left side of the console, select **Troubleshooting**.
3. From the task bar on the left side of the console, select **Diagnostic Provider**.
4. From the task bar on the left side of the console, select **Configuration Data**.
5. Either directly enter a Diagnostic Provider ID in the **Quick link using diagnostic provider ID** text box, or select a process (cluster / node / server) from the available processes displayed at the bottom of the panel under the section title **Server selection topology**.
6. From the list of available diagnostic providers for the selected process, choose the desired diagnostic provider name. The configuration data for that diagnostic provider appears.

Configuration data quick link or server selection

Use this panel to select a Diagnostic Provider server for viewing run time configuration data.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **Configuration Data**

Quick link using Diagnostic Provider ID

From the Configuration data panel, enter a Diagnostic Provider ID to go directly to the page for the configuration data for the Diagnostic Provider for the specific server.

Server selection topology

Use these folders to select server or cluster for viewing the configuration data for a Diagnostic Provider.

If you choose a cluster, whatever action you choose is performed on *each* server in the cluster.

The enterprise applications section shows you the servers that a particular application is running on. If you select a server from this list, the action is performed on that server, not specifically that application.

Diagnostic Providers (selection)

Use this panel to select a Diagnostic Provider from the selected server or cluster.

The list will contain only Diagnostic Providers registered on the selected server or cluster. Not all Diagnostic Providers register with every server in the cell.

You can follow several navigation paths to view this administrative console page. For example, click **Troubleshooting** then expand **Diagnostic Provider** and click on **Tests**. Under **Server selection topology**, click on a server or cluster name then click on a Diagnostic Provider from the list.

Diagnostic Providers

Choose a diagnostic provider from this list.

The path you chose to get to this panel determines which panel displays next.

- If you chose Troubleshooting > Diagnostic Provider > Tests, you see a panel that lists all of the available tests to run on the Diagnostic Provider.
- If you chose Troubleshooting > Diagnostic Provider > State Data, you see a panel that shows the collected state data for the Diagnostic Provider.
- If you chose Troubleshooting > Diagnostic Provider > Configuration Data, you see a panel that shows the configuration data for the Diagnostic Provider.

Configuration data

Use this panel to view the current configuration data for a Diagnostic Provider on a selected server or cluster. Not necessarily every piece of configuration data appears, but data that can be helpful in problem determination is shown.

You can follow several navigation paths to view this administrative console page. For example, click **Troubleshooting** then expand **Diagnostic Provider** and click on **Configuration data** . Under **Server selection topology**, click on a server or cluster name then click on a Diagnostic Provider from the list.

The attributes show information that has been configured for the Diagnostic Provider. You can use the **Save** button to save the information to a file.

Note: Results from a configuration dump contain names that start with either *startup* or *current*. The *startup* entries represent data that was read in by the component at server startup time. The *current* entries contain data that is current – meaning the value of the attributes in use by the runtime at the time the configuration dump was requested.

Node

This is the node name from where the configuration data was collected.

Server

This is the server name from where the configuration data was collected.

Name

This is the name of the attribute for the configuration data.

Value

This is the value of the configuration data.

Description

This is a description of the configuration data.

Viewing the run time state data or configuring the state data collection specifications for a Diagnostic Provider

Use the administrative console to navigate to the state data that can be used to check the health of a server runtime component, or you can configure the state data to be collected for a server.

Before you begin

You must have sufficient authority to execute the action.

About this task

In the server selection topology section, use the view state data radio button to go to the list of registered diagnostic providers. Use the change state data collection specification radio button to modify the state collection specification for the runtime components for a server. Runtime components that have associated diagnostic providers can include their Diagnostic Provider ID (DPID) in their log entries. If you know the DPID, you can enter it directly in the quick link text box.

Procedure

1. Start the administration console.
2. Select **Troubleshooting**.
3. Select **Diagnostic Provider**.
4. Select **State Data**.
5. Select the **View State Data** radio button to simply look at the state data, or select the **Change state data collection specification** radio button to change the configuration.
6. Either directly enter a Diagnostic Provider ID in the **Quick link using diagnostic provider ID** text box, or select a process (cluster / node / server) from the available processes displayed at the bottom of the panel.
   - If you chose the **View State Data** radio button, a panel listing the available Diagnostic Providers appears. Choose one of the providers by clicking on it. A panel displaying the state data appears.
   - If you chose the **Change state data collection specification** radio button, a panel appears that contains a list of the available Diagnostic Providers and a text entry block. The state collection specification for the selected process is managed from this panel. Select one of the available providers by using the checkbox next to it.

Diagnostic Provider State Collection Specification

The State Collection specification provides a mechanism for indicating what additional data diagnostic providers in the system should retain in cases where this additional data could be useful for problem determination or application tuning.

In normal operation, most components should work optimally and not store any operational data that is not needed. There are times, however, when an administrator or automated tool may want a component to collect more information than normal to help in problem determination. This data could then be exposed through a State dump. The State Collection specification was created as a syntax for indicating what additional data the diagnostic providers in the system should retain.

For the syntax of the `aCollectionSpec` string, refer to the `DiagnosticConfigHome` API documentation. It is basically a semicolon (;) separated list of collection specification clauses which are of the form:

```
<DiagnosticProviderName regexp>:<AttributeId regexp>=[0|1]
```

Where the `DiagnosticProviderName` regular expression will make this clause apply to any Diagnostic Provider Name that matches that regular expression. The `AttributeId regexp` and the boolean value (`0` for off, and `1` for on) are stored in the `DiagnosticConfig` object that each Diagnostic Provider uses. Turning

on or off, and processing the clauses left to right allows relatively complex specification. Any specification that is not explicitly turned `on` is considered to be off. This format is explained further in the following examples.

To turn on tracing for all attributes in the `MyDP` Diagnostic Provider:

```
MyDP:.*=1
```

To turn on tracing for all attributes of all Diagnostic Providers (this will probably impact system performance):

```
.*:.*=1
```

To turn on all tracing for all attributes of all Diagnostic Providers beginning with `ConnMgr` (for example, Data Sources):

```
ConnMgr.*:.*=1
```

This specification turns on special collection attributes in the `MyDP` Diagnostic Provider that begin with the string `PoolInfo`. If, however, the attribute begins with `PoolInfo.Db2Pool`, then the collection is off (because it is read left to right).

```
MyDP:PoolInfo.*=1;MyDP:PoolInfo.Db2Pool.*=0
```

It should be noted that State dumps can return important information even in the case where there is no State Collection Specification turned on for a Diagnostic Provider. Diagnostic Providers frequently have to keep some state information in order to operate. Anything in this category is available in a State dump even if there is no special data collection going on. Using the State Collection Specification may increase the amount of data available.

State Data Quick Link or Server Selection

Use this panel to select a server or cluster to either view collected state data, or to configure state data to collect for a Diagnostic Provider.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **State Data**.

Quick link using Diagnostic Provider ID

Enter a Diagnostic Provider ID to go directly to the view page for the collected state data for the Diagnostic Provider.

Server selection topology

Use these radio buttons and folders to select a specific server or cluster for viewing of state data or configuring the specification of state data.

If you choose a cluster, whatever action you choose is performed on *each* server in the cluster.

The enterprise applications section shows you the servers that a particular application is running on. If you select a server from this list, the action is performed on that server, not specifically that application.

View state data
: Select this radio button to view the state data for a Diagnostic Provider. Then select the cell or cluster you want to work with.

Change state data collection settings
: Select this radio button to configure the state collection specification for a Diagnostic Provider. Then select the cluster or managed server you want to work with.

State data

Use this panel to view the current state data for a Diagnostic Provider on a selected server or cluster.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **State data** > select the View state data radio button and then select a server or cluster name > select a Diagnostic Provider from the list.

The attributes show information that has been collected as part of the enabled state collection specification for the Diagnostic Provider. You can use the **save...** button to save the information to a file

Node
This is the node name from where the state data was collected.

Server
This is the server name from where the state data was collected.

Name
This is the name of the state collection specification used to collect the state data.

Value
This is the value of the state collection specification used to collect the state data.

Description
This is a description of the state collection specification used to collect the state data.

Detailed state specification
Use this panel to view the attributes and descriptions of the Diagnostic Provider that you have selected.

To add attributes, select the checkbox next to your chosen diagnostic provider, then select the **Add to specification** button.

To remove a diagnostic provider's sub-component attribute from the state specification, select the sub-component attribute in the displayed list and then select the **Remove from specification** button.

When you are done adding or removing a diagnostic provider's sub-component attributes, select the **Done** button.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **State data** >select the View state data radio button and then select a server or cluster name > select a Diagnostic Provider from the list.

Attribute
This is the individual state collection specification available for the Diagnostic Provider.

Description
This is the description of the individual state collection specification item.

Change state specification
Use this panel to add a Diagnostic Provider and its attributes to the specification for collecting state data.

To add a diagnostic provider (DP) and *all* of its attributes, select the checkbox next to your chosen DP, then click on the **Add to specification** button. To add only *some* of the DP's attributes, click on the DP name itself in the list, and a new panel where you can perform this task appears.

To put the state specification into affect, select the **Apply** or **OK** button.

To reset the specification to its original state, use the **Reset** button.

To manually enter a state specification, update the text area with the state specification and use the **Update** button.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **State data** > select the Change state data collection specification radio button and then select a server or cluster name > select a Diagnostic Provider from the list.

Name
This is a list of available Diagnostic Providers for the server selected.

Modifying the State Collection Specification from wsadmin scripts
In addition to modifying the State Collection Specification from the administrative console, you can also modify these settings using scripts and the wsadmin tool.

About this task

In doing problem determination, you might want to begin collecting additional data during normal processing. This can be accomplished by modifying the State Collection Specification dynamically. This section illustrates how to do that through the wsadmin tool . This technique can be used to turn on traces, as well as to turn off traces. Depending on the usage pattern of the component, the impact should take affect shortly after it is set.

Procedure
1. Capture the DiagnosticService ObjectName into a variable. Enter

   ```
   set DS [lindex [$AdminControl queryNames name=DiagnosticService,*] 0]
   ```
2. Use this variable to drive the method to set the specification. Enter

   ```
   $AdminControl invoke $DS setStateCollectionSpec "SampleDiagnosticProvider:player.*=1;
   SampleDiagnosticProvider:defense.*=1"
   ```

   The specification is of the form **DiagnosticProviderName:AttributeId=0|1...** (with a semicolon at the end, multiple sub-specifications can be entered similar to the TraceSpec). The DiagnosticProviderName and AttributeId can be proper regular expressions.

Running a self diagnostic on a Diagnostic Provider
You can check the status of server runtime components with predefined tests that can be associated with a Diagnostic Provider. Use the administrative console to access these functions.

Before you begin

You must have sufficient authority to execute the action.

About this task

You can access a list of predefined diagnostic tests that you can use to check the status of a server runtime component. Runtime components that have associated diagnostic providers can include their Diagnostic Provide ID (DPID) in their log entries. If you know the DPID, you can enter it directly in the quick link text box. Otherwise, navigate to the desired process by using the tree view displayed at the bottom of the panel.

Procedure
1. Start the administration console.
2. Select **Troubleshooting**.
3. Select **Diagnostic Provider**.

4. Select **Tests** .
5. Either directly enter a Diagnostic Provider ID in the **Quick link using diagnostic provider ID** text box, or select a process (cluster / node / server) from the available processes displayed in the **Server selection topology** section.
6. Select the desired self diagnostic test.
7. Read the output messages from the self diagnostic test.
8. Select a self diagnostic test message by clicking on it. The console displays a panel with the attributes related to the message you chose.

Tests Quick Link or Server Selection

Use this panel to select a Diagnostic Provider server for diagnostic tests.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **Tests**.

Quick link using Diagnostic Provider ID

Enter a Diagnostic Provider ID to go directly to the list of tests for the Diagnostic Provider.

Server selection topology

Use these folders to select server or cluster for viewing the available tests for a Diagnostic Provider.

If you choose a cluster, whatever action you choose is performed on *each* server in the cluster.

The enterprise applications section shows you the servers that a particular application is running on. If you select a server from this list, the action is performed on that server, not specifically that application.

Test selection

Use this panel to select one of the tests that are available for the chosen Diagnostic Provider on the chosen server or cluster.

You can follow several navigation paths to view this administrative console page. For example, click **Troubleshooting** > **Diagnostic Provider** > **Tests** > select a server or cluster name > select a Diagnostic Provider from the list.

Test identification

Choosing a test ID causes the test to run. Results of the test are shown on the Test Results panel.

Test description

A description of the test available to run on the Diagnostic Provider.

Test Results

Use this panel to see the results from the server or cluster members for the selected test.

You can follow several navigation paths to view this administrative console page. For example, click **Troubleshooting** > **Diagnostic Provider** > **Tests** > select a cluster name > select a Diagnostic Provider from the list > select a Test identification from the list.

Multiple results can be returned from a test from each server. The results are sorted by Node, then by Server, then by Severity. You can page through the messages that are returned.

Server

The name of the server where the test result came back from.

Node

The name of the node where the test result came back from.

Severity
The severity of the result from the test run.

Message
A description of the test result.

The entries in this column are linked to another panel. If you click on a message, you can see additional attributes associated with the message.

Test result details
Use this panel to see additional attributes for the selected test result.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **Tests** > select a cluster name > select a Diagnostic Provider from the list > select a Test identification from the list > select a message.

The attributes show information that helped to diagnose the condition described in the message. You can use the **Save** button to save to a file the attributes and the messages to which they correspond.

Name
The name of the test.

Value
This is the value of the test result.

Description
This is a description of the test.

Chapter 21. Troubleshooting help from IBM

If you are not able to resolve a WebSphere Application Server problem by following the steps described in the troubleshooting topics, 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.

If you cannot resolve your problem by any of the preceding methods:

1. Go to the WebSphere Application Server support page.
2. Expand the Contact Support section.
   - Click "Exchanging information with IBM Tech Support", and follow the information collection instructions provided.
   - Click "Directory of worldwide contacts" to find the appropriate contact information for your geography.

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.

IBM Support Assistant

Use the IBM Support Assistant to organize your problem solving investigations and get the latest troubleshooting tools to analyze everything from logs to memory dumps. The IBM Support Assistant provides access to many of the same troubleshooting tools that our IBM Technical Support teams use when solving problems.

IBM Support Assistant Data Collector

The IBM Support Assistant Data Collector is a utility for running automated, symptom-specific data collection scripts. These scripts gather logs, trace, configuration, system-specific information, and other symptom-specific data that is useful to diagnose various potential problems. You can attach the resultant collections to an IBM Service Request so that you can get help from IBM Support. The IBM Support Assistant Data Collector is included with WebSphere Application Server, and can be run from your $*app_server_root*/bin directory. IBM Support might 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.

Consulting

For complex issues such as 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 website http://www.ibm.com/services/fullservice.html.

Problem determination skills

In a large-scale enterprise system such as the WebSphere Application Server for z/OS environment, diagnosis might require a variety of skills to progress from a symptom to fixing the underlying cause of that symptom.

Because WebSphere Application Server for z/OS exploits many of the qualities and services that are unique to the z/OS operating system, diagnosing system-related problems might require skills in the following areas:
- Parallel sysplex
- TCP/IP
- Security Server (RACF) or the equivalent
- Database systems such as DB2 Universal Database™ for z/OS
- UNIX Systems Services

You can find information for many of these topics in the publications available through the z/OS library website.

Similarly, diagnosing application-related problems might require a variety of skills because of the variety of application components that WebSphere Application Server for z/OS supports. Programmers who diagnose application problems in the WebSphere Application Server for z/OS environment need some familiarity with the following:
- The process of assembling, deploying, installing, and running server applications and clients in the WebSphere Application Server for z/OS environment.
- Various tools such as the WebSphere Application Server for z/OS error log, and the job logs for programs running on z/OS.

Diagnosing and fixing problems: Resources for learning

In addition to the information center, there are several Web-based resources for researching and resolving problems related to the WebSphere Application Server.

The WebSphere Application Server support page

The official site for providing tools and sharing knowledge about WebSphere Application Server problems is the WebSphere Application Server support page: http://www.ibm.com/software/webservers/appserv/support.html. Among the features it provides are:
- A search field for searching the entire support site for documentation and fixes related to a specific exception, error message, or other problem. Use this search function before contacting IBM Support directly.
- *Solve a problem* links take you to specific problems and resolutions documented by WebSphere Application Server technical support personnel.
- The *Download* links provide free WebSphere Application Server maintenance upgrades and problem determination tools.
  - *fixes* are software patches which address specific WebSphere Application Server defects. Selecting a specific defect from the list in the *Fixes by version* page takes you to a description of what problem the fix addresses.
  - Fix packs are bundles of multiple fixes, tested together and released as a maintenance upgrade to WebSphere Application Server. Refresh packs are fix packs that also contain new function. If you select a fix pack from this page, you are taken to a page describing the target platform, WebSphere Application Server prerequisite level, and other related information. Selecting the *fix list* link on that page displays a list of the fixes which the fix pack includes. If you intend to install a fix which is part of a fix pack, it is usually better to upgrade to the complete fix pack rather than to just install the individual fix.

Accessing WebSphere Application Server support page resources

Some resources on the WebSphere Application Server support page are marked with a key icon. To access these resources, you must supply a user ID and password, or register if do not already have an ID. When registering, you are asked for your contract number, which is supplied as part of a WebSphere Application Server purchase.

WebSphere Developer Domain

The Developer Domains are IBM-supported sites for enabling developers to learn about IBM software products and how to use them. They contain resources such as articles, tutorials, and links to newsgroups and user groups. You can reach the WebSphere Developer Domain at http://www7b.software.ibm.com/wsdd/.

The IBM Support page

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

Using IBM Support Assistant

IBM Support Assistant is a free troubleshooting application that helps you research, analyze, and resolve problems using various support features and tools. IBM Support Assistant enables you to find solutions yourself using the same troubleshooting techniques used by the IBM Support team, and it allows you to organize and transfer your troubleshooting efforts between members of your team or to IBM for further support.

About this task

IBM Support Assistant helps you diagnose problems with applications that are deployed to WebSphere Application Server. You can track and organize troubleshooting activity, automate the collection of MustGather files, and use tools that analyze artifacts to understand the root cause of errors.

You can customize IBM Support Assistant by installing the problem determination tools that you need to troubleshoot various problems. For example, when the JVM runs out of memory, application threads hang, the JVM crashes, or if performance declines, you can use analysis tools to diagnose these problems.

Procedure

- Follow the installation instructions on IBM Support Assistant (ISA) website at: IBM Support Assistant.
- Learn about suggested add-on tools for IBM Support Assistant that are recommended for use with WebSphere Application Server at Primary Troubleshooting Tools for WebSphere Application Server

Results

Using the IBM Support Assistant Data Collector

The IBM Support Assistant Data Collector for WebSphere Application Server is a tool you can run to gather data from your application server system for problem determination purposes. It replaces the collector tool, which is deprecated.

About this task

Note: The IBM Support Assistant Data Collector for WebSphere Application Server tool focuses on automatic collection of problem data. It also provides symptom analysis support for the various categories of problems encountered by IBM software products. Information pertinent to a type of problem is collected to help identify the origin of the problem under investigation. The tool assists customers by reducing the amount of time it takes to reproduce a problem with the proper RAS tracing levels set, as well as by reducing the effort required to send the appropriate log information to IBM Support.

Procedure

1. Start the tool. The tool runs in console mode by starting the launch script from the command line.
   - In a Windows environment, run the *app_server_root*/bin/isadc.bat command.
   - In a Linux, AIX®, HP-UX, Solaris, IBM i, or zOS environment, run the *app_server_root*/bin/isadc.sh command.

   Note: You can optionally run the tool from a profile bin directory, instead of the *app_server_root*\bin\ directory: *profile_root*/bin/isadc.bat or *profile_root*/bin/isadc.sh.
2. Run the IBM Support Assistant Data Collector tool with the user ID for which you configured your WebSphere Server instance. Depending on what collection you are running, you are asked for additional information to complete the data collection activities. A script might require additional configuration information, information about the sequence of events leading up to the problem you are dealing with, or for your preferences regarding how it completes the collection.

   At each step, the choices are presented as numbered lists and you input the number of your selection and press the enter key. When input is required, prompts are displayed at which you enter your response and press the enter key. You can find collection details for each WebSphere Application Server problem type in their corresponding MustGather documents.

   The tool has a silent collection capability for recording your responses from a console mode session in a file and uses the file to drive subsequent instances of the same collection script. When running in this mode, you are taken to an ordinary interactive session, where you supply the responses to the script prompts. In addition to influencing the current collection, however, your responses are also saved in the file that you named. When the interactive session completes, you can use this response file to start the same script in the future without the need for explicit user input.

   Use the following command to create a response file that contains the answers to all the questions for a certain run through the data collector:

   - *app_server_root*/bin/isadc.sh -record response_filename

   Use the following command to provide the response file when starting the tool:

   - *app_server_root*/bin/isadc.sh -silent response_filename

   The response file is a plain text file, so you can edit it to change the responses as needed. The file looks very much like a Java properties file, with comments that start with #, and a series of key-value pairs. You can add pauses to a response file using one of the following two keys:

   - PauseScriptTime=X, where *X* is a positive integer representing the number of seconds a script pauses. If anything other than a positive integer is found, an error message is written to both the console and the log, followed by a message telling the user to hit the enter key when they are ready for the script to proceed.
   - PauseScript=*user-defined message* - this string is printed to the console, along with a message to hit the enter key you are ready for the script to proceed.

   When using response files, remember that sensitive information, such as user names and passwords, might be stored in these files. It is important that you manage these files in a manner that prevents unauthorized access to sensitive information. Note that passwords are not encrypted.

   Tip: If you are not able to provide root or administrator access to the user to run the collection scripts, make sure that the user ID has administrator privileges for your WebSphere Server, for example, **startServer**, **stopServer**, and **wsadmin** commands.

   By default, the version of the tool (and the various subcomponents) is printed to the console from which it was launched.
3. Stop the collector tool by typing the quit option in console mode.

IBM service call preparation

When you report a problem to IBM service, you will need to provide as much information as possible to help service personnel quickly resolve the problem.

The information you might need to send depends, in part, on the type of problem you have encountered, and includes the following items:
- Job logs for affected address spaces; for example, the controller and any servant regions that the controller terminated
- Job output for affected address spaces, particularly WebSphere Application Server for z/OS messages that are written to the JESMSGLG data set
- The system log (SYSLOG), another source of WebSphere Application Server for z/OS messages
- WebSphere Application Server for z/OS error log
- The system logrec data set or log stream
- CTRACE external writer data sets
- SVC dumps, CEEDUMPs, or other types of dumps produced because of the problem.
- The affected server's environment file, WAS.env, which is located in the HFS:

  `AppServer/config/cells/cellname/nodes/nodename/servers/servername/was.env`

Additionally, IBM service might request you to:
- Provide a description of the circumstances or scenario under which the error occurs.
- Use the `VERBEXIT BBORDATA` subcommand.
- Reset WebSphere variables that are for use only when directed by IBM service.
- Set WebSphere variable values for the location service daemon address space (same as those for servers, with the prefix “ DAEMON_”).

IPCS VERBEXIT subcommand to display diagnostic data

The interactive problem control system (IPCS) is a tool that provides formatting and analysis support for dumps and traces produced by WebSphere Application Server for z/OS and the applications that it hosts.

IBM service personnel might request that you use the IPCS subcommand `VERBEXIT` with the `BBORDATA` verb name to display dump information for WebSphere Application Server for z/OS. The BBORDATA formatters reside in the `WAS_HOME/lib/ipcs/` directory, which must be copied into a dataset that is in the link list or LPA

Entering `VERBEXIT BBORDATA` results in a display of dump contents that can include the following WebSphere Application Server for z/OS structures:
- Global control blocks
- Address space control blocks
- Task control blocks (TCBs)
- ORB control block

Optional parameters control which of these structures are included in the dump display. If you enter `VERBEXIT BBORDATA` without any optional parameters, the dump display includes only global control block content

To enter `VERBEXIT BBORDATA`, you might use any of the methods for entering IPCS subcommands on z/OS, as described in z/OS MVS IPCS User's Guide, SA22-7596. Use the following syntax: `VERBEXIT BBORDATA [ 'parameter [,parameter]...' ]`

Valid parameters are:
- **`GLOBAL(bgvt-address)`**

  Formats and displays cell-level global vector data for the specified address space. This display includes the following formatted control blocks:
  - BGVT address - z/OS Global Vector table
  - ASR Table and ASR Table entries - Active Server Resposity information
- **`ASID(asid-number)`**

  Formats and displays address space information for the daemon, the controller (region), or the servant (region). This display includes the following formatted control blocks:
  - BACB - z/OS address space control block

- BTRC,TBUFSET,TBUF - z/OS Component trace control blocks
- BOAM,BOAMX - z/OS BOA control blocks
- ACRW queue - Application Controller Work element control blocks
- BTCB queues - z/OS control information

Along with `ASID(asid-number)`, IBM service personnel might direct you to specify one of the following parameters, to include additional information in the dump display:

- **BTCB(btcb_address)**

  Formats and displays the specified BTCB and ORB information for the WebSphere Application Server for z/OS TCB.
- **COMMDATA**

  Formats and displays session information.
- **CONFIG**

  Formats and displays configuration information for the address space.
- **OBJADDR(object_address)** and **OBJTYPE(object_type_ID)**

  Formats and displays information for the specified object of the specified type. IBM service personnel will provide the values for you to supply for these parameters.
- **ORBDATA**

  Formats and displays ORB information.
- **TCB(tcb_address)**

  Formats and displays request summary information for the specified task.
- **TRACEBACK**

  Formats and displays ORB information.

- **SUMMARY**

  Summarizes information from some of the other `BBORDATA` optional parameters. For example, for the `GLOBAL` parameter, specifying `SUMMARY` produces a list of active servers.

Example: Output from the command `ip VERBEXIT BBORDATA 'ASID( xx ) TCB( yyyyyyyy )'`:

```
command ===> ip VERBX BBORDATA 'ASID(xx) TCB(xxxxxxxx)'
******************************** TOP OF DATA ********************************
COMPON=WEBSPHERE Z/OS,COMPID=5655A9801,ISSUER=BBORMCDP,ERRNO=04006006

BBOR0012I Formatting Clssname
 Clssname: 2BE6947E
+0000 D9859496 A385E685 82C39695 A3818995 |RemoteWebContain|.............|
+0010 859900                              |er.            |...          |
BBOR0012I Formatting MethodNm
 MethodNm: 2BE69472
+0000 88A3A397 998598A4 85A2A300 00000000 |httprequest.....|.........?.....|
BBOR0012I Formatting ComRtInf
 ComRtInf: 2BE69212
+0000 89974081 8484997E F94BF5F6 4BF4F24B |ip addr=9.56.42.|..@....~.K..K..K|
+0010 F1F6F840 979699A3 7EF1F0F8 F500     |168 port=1085.  |...@....~.....  |
BBOR0026I GMT Time Request was received into CTL region
 TODCLOCK: 00000000
    04/08/2003 12:58:02.926136
BBOR0026I GMT Time Request was Queued to WLM in CTL region
 TODCLOCK: 00000000
    04/08/2003 12:58:02.926263
BBOR0026I GMT Time Request will be Expired (approximated)
 TODCLOCK: 00000000
    04/08/2003 13:08:01.663032
BBOR0026I GMT Time Request was received into SR region
 TODCLOCK: 00000000
    04/08/2003 12:58:02.927729
```

Trace controls for IBM Support

Use these settings to view or modify your trace settings.

To view or set your trace settings, in the administrative console:

- Click **Environment > WebSphere variables**.
- On the Configuration Tab check for any of these properties in the name field and observe the property settings in the value field.
- To change or set a property, specify the property name in the name field and specify the setting in the value field. You can also describe the setting in the description field on this tab.

Note:

- If you use any level of tracing, including setting the ras_trace_defaultTracingLevel property to `1`, verify that you set the ras_trace_outputLocation property to `BUFFER`.

  When the ras_trace_defaultTracingLevel property is set to `1`, exceptions are written to the trace log, as well as to the ERROR log.
- Set the ras_trace_BufferCount property to `4`, and the ras_trace_BufferSize property to `128`.

  These settings reserve 512KB of storage for the trace buffers, which is the minimum amount of storage that can be used, and reduces memory requirements.
- It is best to trace to CTRACE.

  If you are tracing to SYSPRINT with the ras_trace_defaultTracingLevel property set to `3`, you might experience an almost 100% throughput degradation. If you are tracing to CTRACE, however, you might only experience a 15% degradation in throughput.
- Make sure you disable JRAS tracing.

  To disable JRAS tracing, find the following lines in the trace.dat file that is pointed to by the JVM properties file:

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

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

  Ensure that both lines are set to `disable`, or delete the two lines. If the ras_trace_outputLocation property is set, you might be tracing and not know it.

ras_trace_defaultTracingLevel= *n*

Specifies the default tracing level for the product.

Table 100. Valid values and their meanings. Choose a valid value from the following table:

Value	Description
0	No tracing
1	Exception tracing
2	Basic and exception tracing
3	Detailed tracing, including basic and exception tracing

Use this property, together with the ras_trace_basic and ras_trace_detail properties, to set tracing levels for the product subcomponents.Specifies the default tracing level for the product.

Default: 1

Example:

```
ras_trace_defaultTracingLevel=2
```

ras_trace_basic=n | (n,...)

Specifies tracing overrides for particular subcomponents.

Subcomponents, specified by numbers, receive basic and exception traces. If IBM Support directs you to specify more than one subcomponent, use parentheses and separate the numbers with commas. IBM Support provides the subcomponent numbers and their meanings.

Other parts of the product receive tracing as specified on the ras_trace_defaultTracingLevel variable.

Valid values for this property are:
- 0: RAS
- 1: Common Utilities
- 2: COS/Naming
- 3: COMM
- 4: ORB
- 5: IM
- 6: OTS
- 7: Shasta
- 8: Systems Management
- 9: z/OS Wrappers
- A: Daemon
- B: IR
- C: Test
- D: COS/Query
- E: Security
- F: Externalization
- G: Adapter
- H: Lifecycle
- I: Identity
- J: JRAS (internal tracing--via direction from IBM support)
- K: Reference collections
- L: J2EE
- M: Logging
- N: GlueCode

Default: (no default value)

Example:

```
ras_trace_basic=3
```

ras_trace_detail=n | (n,...)

Specifies tracing overrides for particular subcomponents.

Subcomponents, specified by numbers, receive detailed traces. If IBM Support directs you to specify more than one subcomponent, use parentheses and separate the numbers with commas. IBM Support provides the subcomponent numbers and their meanings.

Other parts of the product receive tracing as specified on the ras_trace_defaultTracingLevel variable.

Valid values for this property are:
- 0: RAS
- 1: Common Utilities
- 2: COS/Naming
- 3: COMM
- 4: ORB
- 5: IM
- 6: OTS
- 7: Shasta
- 8: Systems Management
- 9: OS/390 Wrappers
- A: Daemon

- B: IR
- C: Test
- D: COS/Query
- E: Security
- F: Externalization
- G: Adapter
- H: Lifecycle
- I: Identity
- J: JRAS (internal tracing--via direction from IBM support)
- K: Reference collections
- L: J2EE
- M: Logging
- N: GlueCode

Default: (no default value)

Examples:

```
ras_trace_detail=3
ras_trace_detail=(3,4)
```

ras_trace_specific=n | (n,...)

Specifies tracing overrides for specific product trace points.

Trace points are specified by 8-digit, hexadecimal numbers. If IBM Support directs you to specify more than one trace point, use parentheses and separate the numbers with commas. You also can specify a property name by enclosing the name in single quotes. The value of the is handled as if you had specified that value on the ras_trace_specific property.

Default: (no default value)

Examples:

```
ras_trace_specific=03004020
ras_trace_specific=(03004020,04005010)
ras_trace_specific='xyz'
```

[where xyz is an environment variable name]

```
ras_trace_specific=('xyz','abc',03004021)
```

[where xyz and abc are environment variable names]

ras_trace_exclude_specific=n | (n,...)

Specifies product trace points to exclude from tracing activity.

Trace points are specified by 8-digit, hexadecimal numbers. If IBM Support directs you to specify more than one trace point, use parentheses and separate the numbers with commas. You also can specify a property name by enclosing the name in single quotes. The value of the property is handled as if you had specified that value on the ras_trace_exclude_specific property.

Default: (no default value)

Examples:

```
ras_trace_exclude_specific=03004020
ras_trace_exclude_specific=(03004020,04005010)
ras_trace_exclude_specific='xyz'
```

where *xyz* is a property name]

```
ras_trace_exclude_specific=('xyz','abc',03004021)
```

where *xyz* and *abc* are property names

Dump controls for IBM service

Use these controls to gather information that can be used by IBM service.

ras_minorcode_action= *value*

Determines the default behavior for gathering documentation about system exception minor codes.

CEEDUMP

Captures callback and offsets.

Tip: It takes time for the system to take CEEDUMPs and this may cause transaction timeouts. For instance, if the WebSphere variable transaction_defaultTimeout is set to 30 seconds, your application transaction may time out because processing a CEEDUMP can take longer than 30 seconds. To prevent this from happening, either:

- Increase the transaction timeout value, or
- Code ras_minorcode_action=NODIAGNOSTICDATA and make sure the ras_trace_minorCodeTraceBacks variable is not specified.

TRACEBACK

Captures Language Environment and z/OS UNIX traceback data.

SVCDUMP

Captures an MVS dump (but will not produce a dump in the client).

NODIAGNOSTICDATA

Specifies that no diagnostic data will be collected, even if CEEDUMP, TRACEBACK, or SVCDUMP processing occurs because of another WebSphere variable setting. For example, if you code both of the following variables, traceback processing occurs but none of the traceback data is collected: ras_minorcode_action=NODIAGNOSTICDATA and ras_trace_minorCodeTraceBacks=ALL

Default: NODIAGNOSTICDATA

Example:

```
ras_minorcode_action=SVCDUMP
```

ras_trace_minorCodeTraceBacks= *value*

Enables traceback of system exception minor codes. Values are:

```
ALL|all
```

Enables traceback for all system exception minor codes.

Enables traceback of system exception minor codes. Values are:

- *minor_code* Enables traceback for a specific minor code.

  Example: Type

  ```
  1234
  ```

  for minor code

  ```
  C9C21234
  ```

- *(null value)* The default. This setting will not cause gathering of a traceback.

  Default: (null value)

  Example:

  ```
  ras_trace_minorCodeTraceBacks=all
  ```

Chapter 22. Default behavior for OutOfMemory exceptions

WebSphere Application Server Version 8 Fix Pack 2 and later ships with IBM Java 6 R26 on supported operating systems. Beginning with this version, the default behavior for OutOfMemory (OOM) exceptions has changed.

By default in IBM Java 5 and later, the first four OOM exceptions for the lifetime of a Java process produce a PHD-formatted heap dump, a Java dump file (javacore), and a snap dump file. By default in IBM Java 6 R26 and later, the first OOM for the lifetime of a Java process produces a PHD-formatted heap dump, a Java dump file, a snap dump file, and an operating system dump—core file on Linux, AIX, and IBM i (not to be confused with a javacore), user-mode minidump with full memory on Windows operating systems, and SYSTDUMP on z/OS. The second, third, and fourth OOM exceptions produce only a PHD-formatted heap dump and a Java dump file. Therefore, the change in default behavior is an additional system dump on the first OOM exception.

A system dump is a superset of a PHD heap dump. A system dump also includes memory contents (strings, primitives, variable names, and so on), thread and frame local information, some native memory information, and more. For more information, go to http://www.ibm.com/developerworks/opensource/library/j-memoryanalyzer/index.html. This added information can solve a larger class of problems, provide more general insight into a running JVM, and ultimately reduce the time that it takes to solve a problem. In IBM Java 5 R12 or earlier and IBM Java 6 R9 or earlier, a system dump had to be post-processed using the jextract tool. With recent versions of IBM Java, however, including the one that ships with this new OOM behavior, a system dump can be directly loaded by a DTFJ-capable tool such as the Memory Analyzer Tool without any post-processing for OOMs (just like a PHD heap dump). For crashes, however, jextract should still be used.

The default configuration can be observed with the `-Xdump:what` generic JVM argument. For example, running `$WAS/java/bin/java -version -Xdump:what` produces the following output (some output removed). Note the range option in particular.

```
# java -version -Xdump:what
Registered dump agents
...
-Xdump:system:
    events=systhrow,
    filter=java/lang/OutOfMemoryError,
    label=... core.&Y&m&d.&H% M% S.% pid.% seq.dmp,
    range=1..1,
    priority=999,
    request=exclusive+compact+prepwalk
...
-Xdump:heap:
    events=systhrow,
    filter=java/lang/OutOfMemoryError,
    label=... heapdump.% Y% m% d.% H% M% S.% pid.% seq.phd,
    range=1..4,
    priority=500,
    request=exclusive+compact+prepwalk,
    opts=PHD
...
-Xdump:java:
    events=systhrow,
    filter=java/lang/OutOfMemoryError,
    label=... javacore.% Y% m% d.% H% M% S.% pid.% seq.txt,
    range=1..4,
    priority=400,
    request=exclusive+preempt
...
-Xdump:snap:
    events=systhrow,
```

```
filter=java/lang/OutOfMemoryError,
label=... Snap.% Y% m% d.% H% M% S.% pid.% seq.trc,
range=1..4,
priority=300,
request=serial
```

Note: A blank space was added to this example after each occurrence of the % character to avoid inappropriate conversions of the text in this information center.

This configuration can be changed using the -Xdump (http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/topic/com.ibm.java.doc.diagnostics.60/diag/tools/dumpagents_syntax.html) generic JVM arguments (http://www-01.ibm.com/support/docview.wss?uid=swg21417365):

- To revert to the old behavior on OOM, use:

  ```
  -Xdump:system:none -Xdump:system:events=gpf+abort+traceassert+corruptcache
  ```

- To remove PHD heap dumps on OOM, use:

  ```
  -Xdump:heap:none
  ```

- To remove PHD heap dumps on OOM and take system dumps on not just the first OOM but on the first four:

  ```
  -Xdump:heap:none
  -Xdump:system:none
  -Xdump:system:events=gpf+abort+traceassert+corruptcache
  -Xdump:system:events=systhrow,filter=java/lang/OutOfMemoryError,range=1..4,request=exclusive+compact+prepwalk
  ```

It is critical that users ensure that operating systems are correctly configured to produce untruncated system dumps:

- **AIX:**

  http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/topic/com.ibm.java.doc.diagnostics.60/diag/problem_determination/aix_setup_full_core.html

- **IBM i:**

  http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/topic/com.ibm.java.doc.diagnostics.60/diag/problem_determination/i5os_setup_os.html

- **Linux:**

  http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/topic/com.ibm.java.doc.diagnostics.60/diag/problem_determination/linux_setup.html

- **z/OS:**

  http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/topic/com.ibm.java.doc.diagnostics.60/diag/problem_determination/zos_setup_dumps.htm

System dumps are written to the location specified by the file parameter in -Xdump (see the previous label attribute in -Xdump:what). You should change this to a dedicated partition with sufficient space. For example:

```
-Xdump:system:file=/var/dumps/core.% Y% m% d.% H% M% S.% pid.% seq.dmp
```

Note: A blank space was added to this example after each occurrence of the % character to avoid inappropriate conversions of the text in this information center.

You might also consider using the -Xdiagnosticscollector generic JVM argument, which will print a warning to native_stderr.log if the JVM detects that ulimits are set incorrectly on startup. For more information, go to http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/topic/com.ibm.java.doc.diagnostics.60/diag/tools/verifyingyourjavadiagnosticsconfiguration.html

Chapter 23. Configuring the memory leak policy

The leak detection policy for the WebSphere® Application Server is turned off by default. You can configure a leak detection, prevention, and action policy to accommodate your applications and environment so that potential memory leaks are reported and acted upon. Leak detection, prevention, and proactive fixing provides for protection and resiliency for servers that face persistent out of memory errors. When a classloader memory leak is detected, WebSphere Application Server notifies you with informational messages in the log and by taking JVM heapdumps so that you can troubleshoot the problem. Optionally, you might also choose to have WebSphere Application Server mitigate, and if possible, fix the memory leak using reflection and other techniques.

Before you begin

A common error in Java Platform, Enterprise Edition (Java EE) applications is a classloader memory leak. A memory leak can result from subtle application bugs or a more complex use, such as third party libraries. System resources, such as CPU time due to garbage collection and the Java heap are consumed when a leak is present. A system can become unresponsive even though all other resources are available. Unless a protection and early warning system is built in, the system might remain in this degraded state and ultimately die due to an out of memory error.

Note: You can configure WebSphere Application Server to detect, prevent, and take action, if possible, on classloader memory leaks using the memory leak detection policy. For more information about memory leaks, read the Memory Leaks in Java Platform, Enterprise Edition applications topic.

About this task

The leak detection option is disabled by default. You can use Java virtual machine (JVM) custom properties to adjust the leak policy values, such as, enable and disable leak detection, action, and prevention. These custom properties are only applicable to a stand-alone server or managed application server and not to a node agent, admin agent, job manager, or deployment manager. When the application or the server is shutting down, WebSphere Application Server determines the classloaders that have leaked and are held references to all of associated loaded classes and objects. If a classloader leak is detected, a heapdump or systemdump is taken.

All persistent configurations of this service are completed using JVM custom properties. There are no administrative console panels. At runtime, use the MemoryLeakConfig and MemoryLeakAdmin mbeans for configuration and administration respectively; however, the configuration changes are not persisted until JVM custom properties are configured.

The MemoryLeak service and its mbean are active only in an application server that hosts applications and services requests. This service is not active on a Deployment Manager, node agent, administrative agent, or other servers types like WebSphere proxy server and so on.

Tip: Configure the JVM com.ibm.ws.runtime.component.disableMemoryLeakService custom property to permanently disable the service.

Procedure

1. Create or modify JVM custom properties to enable various aspects of the memory leak service, based on the table that follows. Read Java virtual machine custom properties to change the values of the JVM custom properties.

Table 101. com.ibm.ws.runtime.component.MemoryLeakConfig.detectAppCLLeaks

Information	Value
Name	com.ibm.ws.runtime.component.MemoryLeakConfig.detectAppCLLeaks

Table 101. com.ibm.ws.runtime.component.MemoryLeakConfig.detectAppCLLeaks (continued)

Information	Value
Description	When the server is shutting down or an application stops, WebSphere Application Server determines the classloaders that have leaked and issues warnings and other additional information that aids in debugging the memory leak. See also the Improved classloader leak detection PMR.
Default	false

Table 102. com.ibm.ws.runtime.component.MemoryLeakConfig.clearAppCLLeaks

Information	Value
Name	com.ibm.ws.runtime.component.MemoryLeakConfig.clearAppCLLeaks
Description	Enable proactive classloader leak mediation and fixing. When this property is set to true, WebSphere Application Server mediates on behalf of the application to remedy any classloader leaks that are detected.
Default	false

Table 103. com.ibm.ws.runtime.component.MemoryLeakConfig.preventJreMemoryLeaks

Information	Value
Name	com.ibm.ws.runtime.component.MemoryLeakConfig.preventJreMemoryLeaks
Description	Enable WebSphere Application Server to eliminate certain classes of memory leaks that are caused by the JRE loading singletons on the thread context classloader.
Default	true

Table 104. com.ibm.ws.runtime.component.MemoryLeakConfig.generateHeapDumps

Information	Value
Name	com.ibm.ws.runtime.component.MemoryLeakConfig.generateHeapDumps
Description	Set to true to cause a heapdump to be created when a memory leak is detected .
Default	true

Table 105. com.ibm.ws.runtime.component.MemoryLeakConfig.leakSweeperDelay

Information	Value
Name	com.ibm.ws.runtime.component.MemoryLeakConfig.leakSweeperDelay
Description	Delay after an application or module stops to check for classloader leaks.
Default	10000 (milliseconds)

Table 106. com.ibm.ws.runtime.component.MemoryLeakConfig.monitorSystemApps

Information	Value
Name	com.ibm.ws.runtime.component.MemoryLeakConfig.monitorSystemApps
Description	Set to true to watch for memory leaks on system applications that are shipped by IBM, such as the administrative console.
Default	false

Attention: You can fine tune the leak action policy behavior using the following thread and timer leaks JVM custom properties. The ThreadLocal, thread, timer and static leak custom properties are applicable only if com.ibm.ws.runtime.component.MemoryLeakConfig.clearAppCLLeaks is set to true.

Table 107. com.ibm.ws.runtime.component.MemoryLeakConfig.clearReferencesInterruptThreads. Thread and timer leaks

Information	Value
Name	com.ibm.ws.runtime.component.MemoryLeakConfig.clearReferencesInterruptThreads
Description	Set to true for WebSphere Application Server to attempt to interrupt threads that are started by the web application. Interrupting threads is performed using the Thread.interrupt() method. There is a possibility that the target thread might not respond to the interrupt.
Default	true

Table 108. com.ibm.ws.runtime.component.MemoryLeakConfig.clearReferencesHttpClientKeepAliveThread. Thread and timer leaks

Information	Value
Name	com.ibm.ws.runtime.component.MemoryLeakConfig.clearReferencesHttpClientKeepAliveThread
Description	If an HttpClient keep-alive timer thread is not started by this web application and is still running, WebSphere Application Server changes the context class loader from the current classloader to its parent to prevent a memory leak. The keep-alive timer thread stops on its own when the keep-alive threads all die. However, on a busy system that might not happen for some time.
Default	true

Table 109. com.ibm.ws.runtime.component.MemoryLeakConfig.clearReferencesStopTimerThreads. Thread and timer leaks

Information	Value
Name	com.ibm.ws.runtime.component.MemoryLeakConfig.clearReferencesStopTimerThreads
Description	Set to true for WebSphere Application Server to stop any java.util.TimerThreads that are started by the web application.
Default	false

Table 110. com.ibm.ws.runtime.component.MemoryLeakConfig.jvmThreadGroupNames. Thread and timer leaks

Information	Value
Name	com.ibm.ws.runtime.component.MemoryLeakConfig.jvmThreadGroupNames
Description	List of ThreadGroup names to ignore when scanning for threads that are started by the web application that must be shut down. This list is delineated by underscores.
Default	system__RMI Runtime

Attention: The following static class variable leaks properties apply only if com.ibm.ws.runtime.component.MemoryLeakConfig.clearAppCLLeaks is set to true.

Table 111. com.ibm.ws.runtime.component.MemoryLeakConfig.clearReferencesStatic. Static class variable leaks

Information	Value
Name	com.ibm.ws.runtime.component.MemoryLeakConfig.clearReferencesStatic
Description	Set to true for WebSphere Application Server to attempt to make final fields null from loaded classes when a web application stops, as a work around for garbage collection bugs and application coding errors. Applications without memory leaks using recent JVMs should operate correctly when this option is set to false.
Default	false

Table 112. com.ibm.ws.runtime.component.MemoryLeakConfig.filterPrefixes. Static class variable leaks

Information	Value
Name	com.ibm.ws.runtime.component.MemoryLeakConfig.filterPrefixes
Description	Member attributes starting with these filters are not set to null when clearReferencesStatic is true.
Default	java javax com.ibm org sun com.sun. The list is delineated by spaces.

Attention: The following Threadlocal leaks properties apply only if com.ibm.ws.runtime.component.MemoryLeakConfig.clearAppCLLeaks is set to true.

Attention: The following properties apply only if clearReferencesThreadLocal is set to true.

These JVM custom properties are persisted in the WebSphere Application Server configuration model in the `server.xml` file. The following code snippet displays the persisted leak policy configuration from the *server_home*`/config/cells/nodes/servers/server.xml` file of an unmanaged server:

```
<jvmEntries xmi:id="JavaVirtualMachine_1183122130078" verboseModeClass="true" verboseModeGarbageCollection="true" verboseModeJNI="false"
runHProf="false" hprofArguments="" debugMode="false" debugArgs="-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=7777"
genericJvmArguments="-agentlib:getClasses -Xquickstart  -Xalwaysclassgc" executableJarFileName="" disableJIT="false">
 <systemProperties xmi:id="Property_1317048628648" name="com.ibm.ws.runtime.component.MemoryLeakConfig.detectAppCLLeaks" value="true" />
 <systemProperties xmi:id="Property_1318975518491" name="com.ibm.ws.runtime.component.MemoryLeakConfig.clearAppCLLeaks" value="true" />
 <systemProperties xmi:id="Property_1318955284241" name="com.ibm.ws.runtime.component.MemoryLeakConfig.generateSystemDumps" value="false" />
 <systemProperties xmi:id="Property_1319119976147" name="com.ibm.ws.runtime.component.MemoryLeakConfig.generateHeapDumps" value="true" />
 <systemProperties xmi:id="Property_1317048628649" name="com.ibm.ws.runtime.component.MemoryLeakConfig.monitorSystemApps" value="false" />
</jvmEntries>
```

2. Click **Apply**.
3. Click **OK**.
4. Save the changes. Make sure that a file synchronization is performed before restarting the servers.

5. Restart the Application Server for the changes to take effect.

Example

The Memory Leak policy for WebSphere Application Server can be configured and persisted using JVM custom properties as described in this example. At runtime the memory leak detection, prevention and policy configuration can be changed using the MemoryLeakConfig mbean.

Administration of the Memory leak policy can be carried out using the MemoryLeakAdmin mbean. The leak policy affects how the application server responds to a classloader memory leak when an application or when the server is stopped.

You can adjust the Memory leak policy 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 configure and administer the memory leak policy using wsadmin jacl scripting :

```
# Scripting in JACL

# Get the object name of the MemoryLeak Configiration object  you want to change the values on
set leakConfig [$AdminControl completeObjectName "type=MemoryLeakConfig,*"]
WebSphere:cell=smitaNode03Cell,name=LeakConfig,type=MemoryLeakConfig,node=smitaNode03,process=server1

# Get the object name of the MemoryLeak Administration object  you want to issue operations
set leakAdmin [$AdminControl completeObjectName "type=MemoryLeakAdmin,*"]
WebSphere:cell=smitaNode03Cell,name=LeakAdmin,type=MemoryLeakAdmin,node=smitaNode03,process=server1

# Look at all the attributes of the MemoryLeakConfig mbean
wsadmin>$Help all $leakConfig
Name: WebSphere:cell=smitaNode03Cell,name=LeakConfig,type=MemoryLeakConfig,node=smitaNode03,process=server1
        Description: Information on the management interface of the MBean
        Class name: com.ibm.ws.runtime.component.MemoryLeakConfig

Attribute                       Type                            Access
JvmThreadGroupNames             java.lang.String                RW
FilterPrefixes                  java.lang.String                RW
RenewThreadPoolNames            java.lang.String                RW
DetectAppCLLeaks                boolean                         RW
ClearAppCLLeaks                 boolean                         RW
MonitorSystemApps               boolean                         RW
NoDumps                         boolean                         RW
GenerateHeapDumps               boolean                         RW
GenerateSystemDumps             boolean                         RW
ClearReferencesStatic           boolean                         RW
ClearReferencesInterruptThreads   boolean                       RW
ClearReferencesStopTimerThreads   boolean                       RW
ClearReferencesHttpClientKeepAliveThread  boolean               RW
ClearReferencesThreadLocal      boolean                         RW
LeakSweeperDelay                int                             RW
ThreadPoolRenewalDelayFactor    int                             RW
PreventJreMemoryLeaks           boolean                         RW
LeakConfiguration               java.lang.String                RO

Operation
Notifications
Constructors

# Print the current Memory Leak Policy configuration on the console
wsadmin>$AdminControl getAttribute $leakConfig LeakConfiguration
MemoryLeakConfig [getClass()=class com.ibm.ws.runtime.component.MemoryLeakConfig, hashCode()=37266644
preventJreMemoryLeaks true
detectAppCLLeaks true
         monitorSystemApps false
         leakSweeperDelay 10000
clearAppCLLeaks true
         clearReferencesStopTimerThreads false
         clearReferencesHttpClientKeepAliveThread true
         clearReferencesInterruptThreads true
```

```
                 jvmThreadGroupNames [system, RMI Runtime]
         clearReferencesStatic true
                filterPrefixes [java., javax., com.ibm., org., sun., com.sun]
         clearReferencesThreadLocal true
                 renewThreadPoolNames [WebContainer]
                 threadPoolRenewalDelayFactor 1
noDumps false
         generateHeapDumps true
         generateSystemDumps false

# Change the configuration
wsadmin>$AdminControl setAttribute $leakConfig ThreadPoolRenewalDelayFactor 10
wsadmin>$AdminControl setAttribute $leakConfig ClearReferencesStopTimerThreads true

# See the updated configuration
wsadmin>$AdminControl getAttribute $leakConfig LeakConfiguration
MemoryLeakConfig [getClass()=class com.ibm.ws.runtime.component.MemoryLeakConfig, hashCode()=37266644
preventJreMemoryLeaks true
detectAppCLLeaks true
         monitorSystemApps false
         leakSweeperDelay 10000
clearAppCLLeaks true
         clearReferencesStopTimerThreads true
         clearReferencesHttpClientKeepAliveThread true
         clearReferencesInterruptThreads true
                 jvmThreadGroupNames [system, RMI Runtime]
         clearReferencesStatic true
                filterPrefixes [java., javax., com.ibm., org., sun., com.sun]
         clearReferencesThreadLocal true
                 renewThreadPoolNames [WebContainer]
                 threadPoolRenewalDelayFactor 10
noDumps false
         generateHeapDumps true
         generateSystemDumps false

# Look at all the operations of the MemoryLeakAdmin mbean
wsadmin>$Help all $leakAdmin
Name: WebSphere:cell=smitaNode03Cell,name=LeakAdmin,type=MemoryLeakAdmin,node=smitaNode03,process=server1
        Description: Information on the management interface of the MBean
        Class name: com.ibm.ws.runtime.component.MemoryLeakAdmin
Operation
java.lang.String findLeaks()
java.lang.String fixLeaks()
java.lang.String fixLeaks(java.lang.String)

# Find current classloader memory leaks
wsadmin>$AdminControl invoke $leakAdmin findLeaks
CWMML0028I: The following web applications were stopped (reloaded, undeployed),
but their classes from previous runs are still loaded in memory,
thus causing a memory leak.[[78577.075.724.NWALogging#NWALoggingEJB.jar]].

# Fix ALL current classloader memory leaks
wsadmin>$AdminControl  invoke $leakAdmin fixLeaks
CWMML0036I: Please watch the SystemOut log for results of the fix leak operation.

wsadmin>$AdminControl  invoke $leakAdmin fixLeaks {"78577.075.724.NWALogging#NWALoggingEJB.jar"}
CWMML0036I: Please watch the SystemOut log for results of the fix leak operation.
```

Memory leaks in Java Platform, Enterprise Edition applications

Memory leaks come in various types such as thread and ThreadLocal leaks, ClassLoader leaks, system resource leaks, and connection leaks. Approaches to memory leak detection typically involve examination of Java virtual machine tool Interface (JVMTI) or performance monitoring infrastructure (PMI) counters to watch for slow growth in Java or native heap usage.

Note: WebSphere Application Server Version 8.5 provides a top down pattern-based memory leak detection, prevention, and action by watching for suspect patterns in application code at run time. WebSphere Application Server has some means of protection against memory leaks when stopping

or redeploying applications. If leak detection, prevention and action are enabled, WebSphere Application Server monitors application and module activity performs diagnostic actions to detect and fix leaks when an application or an individual module stops. This feature helps in increasing application up time with frequent application redeployments without cycling the server.

Class loader memory leak

Many memory leaks manifest themselves as class loader leaks. A Java class is uniquely identified by its name and the class loader that loaded it. Classes with the same name can be loaded multiple times in a single JVM, each in a different class loader. Each web application gets its own class loader and this is what WebSphere Application Server uses for isolating applications.

An object retains a reference to the class it is an instance of. A class retains a reference to the class loader that loaded it. The class loader retains a reference to every class it loaded. Retaining a reference to a single object from a web application pins every class loaded by the web application. These references often remain after a web application reload. With each reload, more classes are pinned which leads to an out of memory error.

Class loader memory leaks are normally caused by the application code or JRE triggered code.

JRE triggered leaks

Memory leaks occur when Java Runtime Environment (JRE) code uses the context class loader to load an application singleton. These singletons can be threads or other objects that are loaded by the JRE using the context class loader.

If the web application code triggers the initialization of a singleton or a static initializer, the following conditions apply:

- The context class loader becomes the web application class loader.
- A reference is created to the web application class loader. This reference is never garbage collected.
- Pins the class loader, and all the classes loaded by it, in memory.

Application triggered leaks

Categories of application triggered leaks are as follows:

- Custom ThreadLocal class
- Web application class instance as ThreadLocal value
- Web application class instance indirectly held through a ThreadLocal value
- ThreadLocal pseudo-leak
- ContextClassLoader and threads created by web applications
- ContextClassLoader and threads created by classes loaded by the common class loader
- Static class variables
- JDBC driver registration: RMI targets

For more information on application triggered links, see http://wasdynacache.blogspot.com/2012/01/websphere-classloader-memory-leak.html, http://www.websphereusergroup.org.uk/wug/files/presentations/31/Ian_Partridge_-_WUG_classloader_leaks.pdf, and http://www.ibm.com/support/docview.wss?uid=swg1PM39870.

WebSphere Application Server now has some means of protection against memory leaks when stopping or redeploying applications. WebSphere Application Server monitors application and module activity and performs diagnostic actions when an application or an individual module is stopped.

There are three parts to this memory leak feature in WebSphere Application Server: detection, prevention, and action.

1. Detection: Issue warnings when a memory leak is detected. Through a combination of standard API calls and some reflection tricks when a web application is stopped, undeployed or reloaded. WebSphere Application Server checks for known causes of memory leaks and issues warnings when an application leak is detected, as follows:

   ```
   [11/17/11 12:01:05:911 EST] 00000005 LeakDetection E   CWMML0015E: The web application [WasSwat#WasSwatWeb.war]
   created a ThreadLocal with key of type [test.memleak.MyThreadLocal] (value [test.memleak.MyThreadLocal@216c691])
   and a value of type [test.memleak.MyCounter] (value [test.memleak.MyCounter@21942ff]) but failed to remove it
   when the web application was stopped.
   ```

2. Prevention is on by default and applies only to JRE triggered leaks. JRE triggered leaks are prevented by initializing singletons at server startup, when the application Server class loader is the context class loader.

3. Action: Take proactive action to fix memory leaks. These actions have reasonable defaults and are configured on a case-by-case basis.

   ```
   protected void com.ibm.ws.classloader.clearReferences(){
           if(ENABLE_CLEAR_REFERENCES_JDBC)
     clearReferencesJdbc();
    if(ENABLE_CLEAR_REFERENCES_THREADS)
             clearReferencesThreads();
           if(ENABLE_CLEAR_REFERENCES_THREADLOCALS)
     clearReferencesThreadLocals();
    if(ENABLE_CLEAR_REFERENCES_RMI_TARGETS)
             clearReferencesRmiTargets();
           if(ENABLE_CLEAR_REFERENCES_STATICS)
               clearReferencesStaticFinal();
   }
   ```

Table 113. Leak Fix Summary Matrix

Leak cause	How to fix	WebSphere Application Server Java Virtual Machine properties for enabling and controlling
Threadlocal	Renew threads in the threadpool for a configurable period. Getting threads out of the pool enables the threads and threadlocals to be garbage collected.	• com.ibm.ws.runtime.component.MemoryLeakConfig.clearReferencesThreadLocal • com.ibm.ws.runtime.component.MemoryLeakConfig.renewThreadPoolNames • com.ibm.ws.runtime.component.MemoryLeakConfig.threadPoolRenewalDelayFactor • com.ibm.ws.util.ThreadPool.DEFAULT_THREAD_RENEWAL_DELAY
HttpClient keep-alive threads	Switch thread to the parent class loader.	• com.ibm.ws.runtime.component.MemoryLeakConfig.clearReferencesHttpClientKeepAliveThread
Timer threads	Use reflection to stop any new tasks that might be scheduled.	• com.ibm.ws.runtime.component.MemoryLeakConfig.clearReferencesStopTimerThreads
Non JVM controlled threads	If the thread is started using an executor, shut down the executor or interrupt the thread.	• com.ibm.ws.runtime.component.MemoryLeakConfig.clearReferencesInterruptThreads
JDBC drivers	Unregister any JDBC drivers that are registered by the web application that the web application forgot.	• com.ibm.ws.runtime.component.MemoryLeakConfig.preventJreMemoryLeak
ResourceBundle	Clear the ResourceBundle cache of any bundles that were loaded by this class loader or any class loader where this loader is a parent class loader.	• com.ibm.ws.runtime.component.MemoryLeakConfig.preventJreMemoryLeak
RMI targets	Use reflection to clear the values in sun.rmi.transport.ObjectTable.implTable and sun.rmi.transport.ObjectTable.objTable.	• com.ibm.ws.runtime.component.MemoryLeakConfig.preventJreMemoryLeak
Static class variables	WebSphere Application Server nullifies the value of all static class variables of classes that are loaded by the application or module class loader.	• com.ibm.ws.runtime.component.MemoryLeakConfig.clearReferencesStatic com.ibm.ws.runtime.component.MemoryLeakConfig.filterPrefixes

Chapter 24. Collecting Java dumps and core files using the administrative console

You can use the Java runtime environment to create dump and core files to help with troubleshooting. You can use the administrative console to trigger the creation of these dumps and core files.

About this task

The Java virtual machine (JVM) is capable of producing Java dump and core files to aid in troubleshooting. You can use heap dump and system dump files to help you diagnose memory-related problems, such as memory leaks. You can use Java core files to help you diagnose problems where the CPU is persistently 100% busy, when threads are hanging, or where threads are in a deadlock.

Note: The process of generating dump and core files can have a noticeable performance impact on your system that can last for many seconds or minutes. If necessary, use your test and development environments to better understand the impact of generating dump and core files.

Note: Generating a heap dump or a system dump is not supported for a non-IBM JVM. If you attempt to generate a heap dump or a system dump using the administrative console against a non-IBM JVM, the following message is displayed in the administrative console: oror

Procedure

1. In the navigation pane, click **Troubleshooting > Java dumps and cores**.
2. Select the server or servers you need to collect a dump or core from.
3. Click **System Dump** or **Java Core** or **Heap Dump** depending on your need.

Results

The system dump or heap dump or Java core is created and stored in the profile root directory of the server from which you requested the dump or core.

Java dump and core collection

Use this page to generate various Java dumps and cores from within the administrative console.

To view this administrative console page, click **Troubleshooting > Java dumps and cores**.

Select one or more of the listed servers, and click one of the following buttons to generate dumps. You can analyze dumps and cores using problem determination tools available for the IBM Support Assistant.

Note: Generating a heap dump or a system dump is not supported for a non-IBM Java virtual machine (JVM). If you attempt to generate a heap dump or a system dump using the administrative console against a non-IBM JVM, the following message is displayed in the administrative console: oror

Heap dump
A heap dump is a snapshot of JVM memory. It shows live objects in the memory and references between them.

You can use this option to debug conditions such as memory leaks, heap fragmentation or to investigate what objects are consuming the largest part of the memory.

Java core
Use this button to investigate why a server is hanging or investigate messages in the logs that indicate a thread has not completed its work in the expected amount of time.

System dump

Use this button to generate system native dumps of the server process. These dumps can be quite large.

Chapter 25. Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories imply specific default directory locations. This article describes the conventions in use for WebSphere Application Server.

Default product locations - z/OS

app_server_root
: Refers to the top directory for a WebSphere Application Server node.

  The node may be of any type—application server, deployment manager, or unmanaged for example. Each node has its own *app_server_root*. Corresponding product variables are *was.install.root* and *WAS_HOME*.

  The default varies based on node type. Common defaults are *configuration_root*/AppServer and *configuration_root*/DeploymentManager.

configuration_root
: Refers to the mount point for the configuration file system (formerly, the configuration HFS) in WebSphere Application Server for z/OS.

  The *configuration_root* contains the various *app_server_root* directories and certain symbolic links associated with them. Each different node type under the *configuration_root* requires its own cataloged procedures under z/OS.

  The default is /wasv8config/*cell_name*/*node_name*.

plug-ins_root
: Refers to the installation root directory for Web Server Plug-ins.

profile_root
: Refers to the home directory for a particular instantiated WebSphere Application Server profile.

  Corresponding product variables are *server.root* and *user.install.root.*

  In general, this is the same as *app_server_root*/profiles/*profile_name*. On z/OS, this will always be *app_server_root*/profiles/default because only the profile name "default" is used in WebSphere Application Server for z/OS.

smpe_root
: Refers to the root directory for product code installed with SMP/E or IBM Installation Manager.

  The corresponding product variable is *smpe.install.root*.

  The default is /usr/lpp/zWebSphere/V8R5.

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.

APACHE INFORMATION. This information may include all or portions of information which IBM obtained under the terms and conditions of the Apache License Version 2.0, January 2004. The information may also consist of voluntary contributions made by many individuals to the Apache Software Foundation. For more information on the Apache Software Foundation, please see http://www.apache.org. You may obtain a copy of the Apache License at http://www.apache.org/licenses/LICENSE-2.0.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site (www.ibm.com/legal/copytrade.shtml).

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

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

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

Index