

MQSeries<sup>®</sup> Adapter Kernel for Multiplatforms



# Problem Determination Guide

*Version 1 Release 1*

**Note:** Before using this information and the product it supports, read the information in “Notices” on page 29.

**First Edition (December 2000)**

This edition applies to version 1, release 1 of MQSeries Adapter Kernel for Multiplatforms (product number 5648-D75) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You can make comments on this information via e-mail at [idrcf@hursley.ibm.com](mailto:idrcf@hursley.ibm.com).

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

© **Copyright International Business Machines Corporation 2000. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Figures.</b> . . . . .	<b>v</b>	Trace errors involving socket handlers . . . . .	23
<b>Tables.</b> . . . . .	<b>vii</b>	Instantiation of trace client fails . . . . .	23
<b>About this book</b> . . . . .	<b>ix</b>	Trace clients or servers loop. . . . .	23
Who should read this book . . . . .	ix	Problems with MQSeries. . . . .	23
Document organization . . . . .	ix	Queue not found . . . . .	23
Related information . . . . .	ix	Channel down . . . . .	23
Conventions used in this book . . . . .	x	Queue depth exceeded . . . . .	24
<b>Chapter 1. Tracing</b> . . . . .	<b>1</b>	Logs exceed available space. . . . .	24
Introduction to tracing . . . . .	1	MQSeries JAR files not found . . . . .	24
Trace destinations . . . . .	2	Unable to load message catalog . . . . .	24
Configuration . . . . .	2	Problems with MQSeries Integrator . . . . .	24
Trace levels . . . . .	3	Message not understood . . . . .	24
Configuring tracing . . . . .	4	Receiving messages from MQSI . . . . .	24
Defining trace clients . . . . .	4	Problems with JMS. . . . .	25
Defining trace servers . . . . .	11	JNDI initialization fails . . . . .	25
Reading trace messages . . . . .	18	JMS communication mode fails . . . . .	25
<b>Chapter 2. Common configuration problems</b> . . . . .	<b>21</b>	Errors locating QueueConnectionFactory or Queue objects . . . . .	25
Kernel configuration problems . . . . .	21	Binding problem on HP-UX. . . . .	26
File aqmsetup not found . . . . .	21	Code-page conversion problem on HP-UX . . . . .	26
File aqmconfig.xml not found . . . . .	21	Miscellaneous problems . . . . .	26
No response to messages. . . . .	21	Display errors on UNIX . . . . .	26
Target adapter not found. . . . .	21	Core dump on Solaris. . . . .	26
Error parsing body data . . . . .	22	Segmentation violation on AIX. . . . .	26
Exceptions from native adapters . . . . .	22	Java memory problem: OutOfMemoryError . . . . .	27
Adapter daemon shuts down . . . . .	22	XML parser problems. . . . .	27
Messages received but not sent to applications . . . . .	22	Using WebSphere . . . . .	27
Messages not removed from incoming queues . . . . .	23	<b>Notices</b> . . . . .	<b>29</b>
		Trademarks . . . . .	31
		<b>Index</b> . . . . .	<b>33</b>



---

## Figures

1. Setting trace-configuration values for an application . . . . .	4	11. The structure of the default TraceServer definition . . . . .	12
2. The structure of the default TraceClient definition . . . . .	5	12. The default TraceServer definition: message catalog . . . . .	13
3. The default TraceClient definition: message catalog . . . . .	6	13. The default TraceServer definition: message buffering . . . . .	13
4. The default TraceClient definition: message buffering . . . . .	6	14. The default TraceServer definition: handler registration. . . . .	14
5. The default TraceClient definition: handler registration . . . . .	7	15. Configuring a multiframe handler in a trace server . . . . .	15
6. TraceClient definition: multiple handler registrations . . . . .	7	16. Configuring a console handler for a trace server . . . . .	16
7. Configuring a console handler for a trace client . . . . .	8	17. Configuring trace workers for socket handlers . . . . .	17
8. Configuring a file handler for a trace client . . . . .	9	18. Configuring trace workers for ENA handlers . . . . .	17
9. Configuring socket and ENA handlers for a trace client . . . . .	10	19. The complete default TraceServer definition . . . . .	18
10. The complete default TraceClient definition . . . . .	11	20. Two trace messages. . . . .	19



---

## Tables

1. Conventions used in this book . . . . x





---

## About this book

This document describes tools for solving specific problems with the MQSeries Adapter Kernel. It contains a set of common symptoms of problems and their possible solutions, and it describes the use of tracing to identify less common problems.

---

## Who should read this book

This document is written for developers, administrators, and users of the MQSeries Adapter Kernel. It is assumed that readers are familiar with the MQSeries Adapter Kernel and the MQSeries Adapter Builder tool.

---

## Document organization

This document is organized as follows:

- “Chapter 1. Tracing” on page 1 describes how to configure and use tracing for the MQSeries Adapter Kernel.
- “Chapter 2. Common configuration problems” on page 21 describes some common configuration problems and offers possible solutions.

---

## Related information

For further information on the topics discussed in this manual, see the following documents:

- *Quick Beginnings*

---

## Conventions used in this book

MQSeries Adapter Kernel documentation uses the following typographical and keying conventions.

Table 1. Conventions used in this book

Convention	Meaning
<b>Bold</b>	Indicates command names. When referring to graphical user interfaces (GUIs), indicates menus, menu items, labels, and buttons.
Monospace	Indicates text you must enter at a command prompt and values you must use literally, such as file names, paths, and elements of programming languages such as functions, classes, and methods. Monospace also indicates screen text and code examples.
<i>Italics</i>	Indicates variable values you must provide (for example, you supply the name of a file for <i>fileName</i> ). Italics also indicates emphasis and the titles of books.
%	Represents the UNIX command-shell prompt for a command that does not require <b>root</b> privileges.
#	Represents the UNIX command-shell prompt for a command that requires <b>root</b> privileges.
C:\>	Represents the command prompts on Windows® systems.
>	When used to describe a menu, shows a series of menu selections. For example, “Click <b>File</b> > <b>New</b> ” means “From the <b>File</b> menu, click the <b>New</b> command.”
Entering commands	When instructed to “enter” or “issue” a command, type the command and then press Return. For example, the instruction “Enter the <b>ls</b> command” means type <b>ls</b> at a command prompt and then press Return.
[ ]	Enclose optional items in syntax descriptions.
{ }	Enclose lists from which you must choose an item in syntax descriptions.
	Separates items in a list of choices enclosed in braces ( { } ) in syntax descriptions.
...	Ellipses in syntax descriptions indicate that you can repeat the preceding item one or more times. Ellipses in examples indicate that information was omitted from the example for the sake of brevity.

---

## Chapter 1. Tracing

Tracing is the general process of collecting detailed information about how a process runs. Trace information can be used to debug an application or to diagnose problems within a product. In the case of MQSeries Adapter Kernel, tracing can be used to collect information about application-specific adapters or about the kernel itself. Tracing can assist both application developers and system administrators, and it is often crucial when asking for product support.

Modern software products like MQSeries Adapter Offering typically come with a built-in capacity to collect trace information. Enabling and collecting trace is simply a matter of setting run-time configuration parameters; these parameters direct the product to capture trace information and put it somewhere for collection and analysis.

Trace information is collected in the form of a set of *trace messages*. The number of trace messages is determined by the *trace level*, which specifies the kinds of information to capture. Collected messages are written to one or more destinations. Examples of destinations include a console window and a named file.

See “Configuring tracing” on page 4 for information on configuring tracing.

See “Reading trace messages” on page 18 for information on interpreting trace messages.

---

### Introduction to tracing

A trace message contains the state of message processing at a certain point within the kernel. You can use trace messages to help diagnose problems with the kernel or with your adapters.

MQSeries Adapter Offering uses the following components to handle tracing:

- A trace client.
- A trace server, a daemon that retrieves the trace messages by using either a socket or native adapter and passes the trace messages to the trace worker.
- A trace worker, a special trace client. A trace worker provides additional methods for use by the trace server, and it operates under defaults different from those used by the standard trace client.
- Output handlers, which determine how and where trace messages are written. Each trace client is assigned a set of handlers.

Multiple instances of a trace client can exist for each processing thread. However, within the kernel, only one instance of a trace client is used for each combination of source or destination logical identifier and thread. All kernel processes using a thread write to the trace client for that thread. Each trace message contains an identifier for the thread, so you can follow all trace messages from a specific thread. To trace a message flow from end to end, you must manually examine and correlate trace messages from both sides of the kernel.

## Trace destinations

Each trace client can write its trace messages to one or more of the following:

- The console (default).
- One or more files.
- A trace server, through a socket connection.
- A trace server, by putting the message onto a queue.

Each of these destinations is managed by a handler associated with the trace client. Trace clients writing to windows, files, and remote servers over sockets use console, file, and socket handlers, respectively. A trace client that puts its messages onto a queue for delivery to a remote server uses a handler for an Adapter Kernel built-in native adapter; this is called a native-adapter (ENA) handler. Use of socket and ENA handlers also requires the use of trace servers.

---

## Configuration

Configuration for all aspects of an MQSeries Adapter Kernel application, including tracing, is contained in an XML configuration file. When installed, the kernel provides a directory called `samples`, which contains a default configuration file, `aqmconfig.xml`, and a configuration script, `aqmsetup`. New applications can copy and modify these files to suit their needs, or new ones can be created. The Adapter Kernel looks at the environment variable `AQMSETUPFILE` to determine the setup script to use; the setup script indicates the directory containing the appropriate configuration file.

The file contains configuration information for each source logical identifier, destination logical identifier, and adapter daemon. Each of these applications receives an application identifier when it starts. The application identifier acts as a label for configuration entries. Each application looks up its configuration, including tracing, by looking for its application identifier in the file.

Each application has three trace-related settings:

- Whether tracing is enabled or disabled.
- The trace level (see “Trace levels” on page 3).

- The trace client's application identifier.

These values are stored between pairs of XML tags. (For general information on configuring applications, see *Quick Beginnings*.)

In addition to adapters, trace clients and servers have application identifiers in the configuration file. For example, the configuration under a trace client's application identifier specifies the handlers to use. As a result of abstracting this information into entries for trace clients and servers, many source and destination logical identifiers and adapter daemons can use the same trace configuration while maintaining the flexibility to turn their own tracing on or off. Trace servers, which are used by socket and native-adapter handlers, are also configured under application identifiers.

## Trace levels

MQSeries Adapter Offering defines the following trace levels:

0	No trace messages
1	Information only
2	Warning only
3	Information and warning
4	Error only
7	Information, warning, and error
128	Method entry
256	Method exit
384	Method entry and exit
512	Exceptions
903	Information, warning, error, exceptions, entry, and exit
-1	All possible messages

Several of the defined levels are combinations of other levels; for example, the trace level to collect both method entry and exit events (384) is the sum of the trace levels for method entry (128) and method exit (256). You can set any combination by summing the corresponding trace levels. For example, to trace warnings (2) and errors (4), specify trace level 6.

---

## Configuring tracing

To configure tracing for an application, you add trace information to the application's entry in the XML configuration file. The trace-related information includes:

- Whether or not to enable tracing.
- The level of tracing to use when trace is enabled.
- Optionally, an application identifier for the trace client. If you do not specify this identifier, the default trace client identifier, `TraceClient`, is used.

**Note:** XML is case-sensitive, so the names of the XML elements and attributes used in tags must capitalize the correct letters. The names of some elements and attributes differ only in use of capitalization.

To set trace values for an application, use the following XML elements within the application's definition element (the `ePICApplication` element):

- `epictrace`: Enables or disables tracing. Set to `true` or `false`.
- `epictracelevel`: Sets the trace level. See "Trace levels" on page 3 for the predefined trace levels.
- `epictraceclientid`: Identifies the trace-client configuration to use. The default value is `TraceClient`.

The example shows the setting of trace values for an application with application identifier (`ePICAppID`) `TEST1`. Tracing is enabled, the trace level is 6, and the trace-client identifier is the default `TraceClient`.

```
<Epic o="Epic">
  <ePICApplications o="ePICApplications">
    <ePICApplication ePICAppID="TEST1">
      <epictrace>true</epictrace>
      <epictracelevel>6</epictracelevel>
      <epictraceclientid>TraceClient</epictraceclientid>
      ...
    </ePICApplication>
  </ePICApplications>
</Epic>
```

*Figure 1. Setting trace-configuration values for an application*

You can also define your own trace clients, which can then be referenced by `epictraceclientid` elements.

### Defining trace clients

The default `aqmconfig.xml` file contains an entry for the default trace-client identifier, `TraceClient`. This entry reflects the default values for `TraceClient`. It can be used as a template for configuring new trace clients, or it can be

modified, which effectively changes the behavior of TraceClient. If you modify the definition of TraceClient, applications that specify no trace-client identifier pick up the new values for TraceClient.

The default values for TraceClient are built into the adapter kernel. You can use the definition in the configuration file to override the default behavior, but you cannot eliminate the defaults by removing the TraceClient definition from the configuration file. If you delete the TraceClient definition, applications that specify no trace-client identifier use the built-in values.

Additional trace clients can be defined, and the TraceClient definition provides a model. Figure 2 shows the structure of a configuration entry for a trace client. Trace clients are defined in the configuration file as additional applications by using the ePICAApplication element. Trace clients are distinguished by name, specified in the epicappid attribute. The configuration details are specified inside the application's ePICTraceExtensions element; the element's single attribute takes a fixed value, cn="epicappextensions".

```
<ePICAApplication epicappid="TraceClient">
  <ePICTraceExtensions cn="epicappextensions">
    ...
  </ePICTraceExtensions>
</ePICAApplication>
```

Figure 2. The structure of the default TraceClient definition

### Message file for trace clients

Internally, trace messages are handled as numeric codes, or *message identifiers*. When the kernel writes trace messages, the internal function that generates the message uses a message catalog file to retrieve the string corresponding to the message identifier. This allows message catalogs to be translated into multiple languages, so applications can choose a preferred language. Figure 20 on page 19 illustrates message codes and corresponding message strings.

The epictracemessagefile element names the message catalog to use if none is explicitly passed to the message-generation function. The definition of TraceClient explicitly names the default catalog, com.ibm.epic.trace.client.TraceMessage. Other trace clients can omit the epictracemessagefile element if the default file is acceptable.

```

<ePICAApplication epicappid="TraceClient">
  <ePICTraceExtensions cn="epicappextensions">
    <epictracemessagefile>
      com.ibm.epic.trace.client.TraceMessage
    </epictracemessagefile>
    ...
  </ePICTraceExtensions>
</ePICAApplication>

```

Figure 3. The default TraceClient definition: message catalog

### Message buffering for trace clients

When trace messages are written, they can be written synchronously or asynchronously. The `epictracesyncoperation` element, which is passed to the trace handler, controls this behavior. Use a value of `true` for synchronous behavior or `false`—the default—for asynchronous behavior. The definition of `TraceClient` explicitly requests asynchronous behavior. Other trace clients can omit the `epictracesyncoperation` element if asynchronous writing is acceptable.

```

<ePICAApplication epicappid="TraceClient">
  <ePICTraceExtensions cn="epicappextensions">
    ...
    <epictracesyncoperation>false</epictracesyncoperation>
    ...
  </ePICTraceExtensions>
</ePICAApplication>

```

Figure 4. The default TraceClient definition: message buffering

### Message handlers for trace clients

The message handlers registered for a trace client specify where the collected trace messages are sent. The four possible destinations are:

- The console
- A file
- A trace server, through a socket connection
- A trace server, through a native adapter

A different handler manages each output destination. For trace clients, the possible handlers are:

- `com.ibm.logging.ConsoleHandler`
- `com.ibm.logging.FileHandler` (a single file)
- `com.ibm.logging.MultiFileHandler` (a backed-up circular file)
- `com.ibm.logging.SocketHandler`



- `com.ibm.epic.trace.client.ENAHandler`

You register a handler by using the `epictracehandler` element. The `TraceClient` definition sends trace messages to the console by default. Other trace clients must specify one or more handlers. Each registered handler must also be configured; see “Configuring handlers” for more information.

```
<ePICAApplication epicappid="TraceClient">
  <ePICTraceExtensions cn="epicappextensions">
    ...
    <epictracehandler>com.ibm.logging.ConsoleHandler</epictracehandler>
    ...
  </ePICTraceExtensions>
</ePICAApplication>
```

*Figure 5. The default TraceClient definition: handler registration*

To specify multiple handlers, use `Value` elements within the `epictracehandler` element. For example, Figure 6 illustrates how to register the file handler in addition to the console handler for `TraceClient`.

```
<ePICAApplication epicappid="TraceClient">
  <ePICTraceExtensions cn="epicappextensions">
    ...
    <epictracehandler>
      <Value>com.ibm.logging.ConsoleHandler</Value>
      <Value>com.ibm.logging.FileHandler</Value>
    </epictracehandler>
    ...
  </ePICTraceExtensions>
</ePICAApplication>
```

*Figure 6. TraceClient definition: multiple handler registrations*

**Configuring handlers:** Each of the five handlers has different requirements, so each needs its own configuration. For example, each of the handlers needs to know how to format its output. A file handler also needs to know to which file it writes. The `TraceClient` entry in the `aqmconfig.xml` file illustrates the configuration of each possible type of handler, but other trace clients need to configure only the handlers they register.

The `epictracehandler` element is used to register the handlers to be used by the trace client. The registered handlers must also be configured. The `ePICTraceHandler` element is used to configure individual handlers. This element uses the attribute `epictracehandler` to indicate the handler being configured; this allows multiple handlers to be configured. If no configuration values are entered for a specific handler, the handler uses default values. Handlers that are not currently registered can be configured; they simply are not used until they are also registered.

*Console handler:* A trace client registering the console handler can also indicate the formatter to be used by the handler. MQSeries Adapter Kernel provides the class `com.ibm.epic.trace.client.EpicTraceFormatter` for use with console and file handlers, and this formatter is the default. Console handlers require no further configuration. If specified, the formatter is indicated within an `epictraceformatter` element.

```
<ePICApplication epicappid="TraceClient">
  <ePICTraceExtensions cn="epicappextensions">
    ...
    <epictracehandler>com.ibm.logging.ConsoleHandler</epictracehandler>
    <ePICTraceHandler epictracehandler="com.ibm.logging.ConsoleHandler">
      <epictraceformatter>
        com.ibm.epic.trace.client.EpicTraceFormatter
      </epictraceformatter>
    </ePICTraceHandler>
    ...
  </ePICTraceExtensions>
</ePICApplication>
```

*Figure 7. Configuring a console handler for a trace client*

*File handler:* A trace client registering the file handler can indicate the formatter to be used by the handler and the file to hold the generated messages. MQSeries Adapter Kernel provides the class `com.ibm.epic.trace.client.EpicTraceFormatter` for use with console and file handlers, and this formatter is the default. The file name can be anything; the default file name for the `TraceClient` is `trc.log`. Note that `TraceClient` does not write to a file unless the file handler is registered in the `epictracehandler` element.

The formatter can be explicitly specified within an `epictraceformatter` element; the file is specified within an `epictracefilename` element. This handler generates one file, which can grow indefinitely in size.

```

<ePICAApplication epicappid="TraceClient">
  <ePICTraceExtensions cn="epicappextensions">
    ...
    <epictracehandler>com.ibm.logging.FileHandler</epictracehandler>
    ...
    <ePICTraceHandler epictracehandler="com.ibm.logging.FileHandler">
      <epictraceformatter>
        com.ibm.epic.trace.client.EpicTraceFormatter
      </epictraceformatter>
      <epictracefilename>trc.log</epictracefilename>
    </ePICTraceHandler>
    ...
  </ePICTraceExtensions>
</ePICAApplication>

```

Figure 8. Configuring a file handler for a trace client

*Multifile handler::* See “Multifile handler” on page 14 for more information on configuring a multifile handler.

*Socket and ENA Handlers:* A trace client registering either a socket or ENA handler can indicate the formatter to be used by the handler and application identifier of the destination trace server. MQSeries Adapter Kernel provides the class `com.ibm.epic.trace.client.EpicXMLFormatter` for use with socket and ENA handlers, and this formatter is the default. The formatter can be explicitly specified within an `epictraceformatter` element, as with console and file handlers.

Socket and ENA handlers forward trace messages to a trace server through a socket connection or a built-in native adapter. These trace servers are also configured under application identifiers. Both socket and ENA handlers must indicate which trace server to use. The trace server is specified with the `epicdepappid` (dependency application) element. This element is not nested within `ePICTraceHandler` elements; it occurs immediately within the `ePICTraceExtensions` element. MQSeries Adapter Kernel comes with a default trace server, `TraceServer`. If no trace server is specified, the default `TraceServer` name is used.

Note that `TraceClient` does not use a trace server unless a socket or ENA handler is registered in the `epictracehandler` element.

The trace server named in the `epicdepappid` element, unless it is the default `TraceServer`, must also be configured within the configuration file. The trace-server application identifier (used in the `epicdepappid` element) is used by a socket handler to obtain the host name and port number for connecting to the trace server. If the defaults are not being used for the socket handler,

the entries for the socket handler also need to be configured when defining the trace server. (See “Defining trace servers” on page 11 for more information.)

An ENA handler uses a native adapter for sending trace messages to the trace server. The trace-server application identifier (used in the `epicdepappid` element) is used by the ENA handler as the destination application identifier. The configuration entry for the trace server must include the appropriate communications information within the `AdapterRouting` element for the native adapter. (See “Defining trace servers” on page 11 for more information.)

```
<ePICAApplication epicappid="TraceClient">
  <ePICTraceExtensions cn="epicappextensions">
    ...
    <epictracetracehandler>com.ibm.logging.ConsoleHandler</epictracetracehandler>
    ...
    <epicdepappid>TraceServer</epicdepappid>
    <ePICTraceHandler epictracetracehandler="com.ibm.epic.trace.client.ENAHandler">
      <epictracetraceformatter>
        com.ibm.epic.trace.client.EpicXMLFormatter
      </epictracetraceformatter>
    </ePICTraceHandler>
    <ePICTraceHandler epictracetracehandler="com.ibm.logging.SocketHandler">
      <epictracetraceformatter>
        com.ibm.epic.trace.client.EpicXMLFormatter
      </epictracetraceformatter>
    </ePICTraceHandler>
  </ePICTraceExtensions>
</ePICAApplication>
```

*Figure 9. Configuring socket and ENA handlers for a trace client*

The complete configuration entry for the `TraceClient` application is shown in Figure 10 on page 11. This client registers only the console handler, but the entry also illustrates the configuration of file, socket and ENA handlers. Those handlers are not used unless they are registered instead of, or in addition to, the console handler.

```

<ePICApplication epicappid="TraceClient">
  <ePICTraceExtensions cn="epicappextensions">
    <epictracemessagefile>
      com.ibm.epic.trace.client.TraceMessage
    </epictracemessagefile>
    <epictracesyncoperation>>false</epictracesyncoperation>
    <epictracehandler>com.ibm.logging.ConsoleHandler</epictracehandler>
    <ePICTraceHandler epictracehandler="com.ibm.logging.ConsoleHandler">
      <epictraceformatter>
        com.ibm.epic.trace.client.EpicTraceFormatter
      </epictraceformatter>
    </ePICTraceHandler>
    <ePICTraceHandler epictracehandler="com.ibm.logging.FileHandler">
      <epictraceformatter>
        com.ibm.epic.trace.client.EpicTraceFormatter
      </epictraceformatter>
      <epictracefilename>trc.log</epictracefilename>
    </ePICTraceHandler>
    <epicdepappid>TraceServer</epicdepappid>
    <ePICTraceHandler epictracehandler="com.ibm.epic.trace.client.ENAHandler">
      <epictraceformatter>
        com.ibm.epic.trace.client.EpicXMLFormatter
      </epictraceformatter>
    </ePICTraceHandler>

    <ePICTraceHandler epictracehandler="com.ibm.logging.SocketHandler">
      <epictraceformatter>
        com.ibm.epic.trace.client.EpicXMLFormatter
      </epictraceformatter>
    </ePICTraceHandler>
  </ePICTraceExtensions>
</ePICApplication>

```

Figure 10. The complete default TraceClient definition

## Defining trace servers

When a component of the kernel starts, it reads the configuration file to determine any run-time settings. With respect to tracing, it uses the application identifier it received on startup to determine whether trace is enabled.

If trace is enabled, a trace client is instantiated. Then, based on the application identifier of its instantiating application, the trace client determines whether trace is enabled, the trace level, and the trace client identifier. Based on the trace client identifier, the trace client determines which handlers to use.

If a socket or ENA handler is registered, the trace server must be started with the same application identifier as that used in the trace client's epicdepappid element. The trace server then instantiates a number of special trace clients,

called trace workers. The configuration for the trace workers is embedded in that of the trace server; trace workers do not have configuration entries under separate application identifiers.

The default `aqmconfig.xml` file contains an entry for the default trace-server identifier, `TraceServer`. This entry reflects the default values for `TraceServer`. It can be used as a template for configuring new trace servers, or it can be modified, which effectively changes the behavior of `TraceServer`.

The default values for `TraceServer` are built into the adapter kernel. You can use the definition in the configuration file to override the default behavior, but you cannot eliminate the defaults by removing the `TraceServer` definition from the configuration file. If you delete the `TraceServer` definition, applications that specify no trace-server identifier use the built-in values.

Like trace clients, trace servers are configured within `ePICApplication` elements, and the attribute `epiccappid` specifies the application identifier. The body of the configuration entry consists of two elements nested within the `ePICApplication` element, an `AdapterRouting` element, and an `ePICTraceExtensions` element. Trace servers have `AdapterRouting` elements when they are using ENA handlers, because an ENA handler uses a native adapter, which obtains its configuration values from the `AdapterRouting` element. This discussion concentrates on the `ePICTraceExtensions` element. For information on creating the `AdapterRouting` element, see *Quick Beginnings*.

```
<ePICApplication epiccappid="TraceServer">
  <AdapterRouting cn="epicadapterrouting">
    <epicmqppqueueemgr>DEFAULT</epicmqppqueueemgr>
    <ePICBodyCategory epicbodycategory="DEFAULT">
      <ePICBodyType epicbodytype="DEFAULT">
        <epicreceivemode>MQPP</epicreceivemode>
        <epicreceivemqppqueue>TraceServerAIQ</epicreceivemqppqueue>
      </ePICBodyType>
    </ePICBodyCategory>
  </AdapterRouting>
  <ePICTraceExtensions cn="epicappextensions">
    ...
  </ePICTraceExtensions>
</ePICApplication>
```

Figure 11. The structure of the default `TraceServer` definition

### Message file for trace servers

Like trace clients, trace servers use the `epictracemessagefile` element to provide a message catalog to the internal function that generates message strings. (For more information, see “Message file for trace clients” on page 5.) The definition of `TraceServer` explicitly names the default catalog,

com.ibm.epic.trace.server.TraceServerMessage. Other trace servers can omit the epictracemessagefile element if the default file is acceptable.

```
<ePICAApplication epicappid="TraceServer">
  <AdapterRouting cn="epicadapterrouting">
    ...
  </AdapterRouting>
  <ePICTraceExtensions cn="epicappextensions">
    <epictracemessagefile>
      com.ibm.epic.trace.server.TraceServerMessage
    </epictracemessagefile>
    ...
  </ePICTraceExtensions>
</ePICAApplication>
```

Figure 12. The default TraceServer definition: message catalog

### Message buffering for trace servers

Like trace clients, trace servers use the epictracesyncoperation element to indicate if messages are to be written synchronously or asynchronously. The default value, false, specifies asynchronous behavior. (For more information, see “Message buffering for trace clients” on page 6.) The definition of TraceServer explicitly requests asynchronous behavior. Other trace servers can omit the epictracesyncoperation element if asynchronous writing is acceptable.

```
<ePICAApplication epicappid="TraceServer">
  <AdapterRouting cn="epicadapterrouting">
    ...
  </AdapterRouting>
  <ePICTraceExtensions cn="epicappextensions">
    ...
    <epictracesyncoperation>false</epictracesyncoperation>
    ...
  </ePICTraceExtensions>
</ePICAApplication>
```

Figure 13. The default TraceServer definition: message buffering

### Message handlers for trace servers

The message handlers registered for a trace server specify where the collected trace messages are sent. The three possible destinations are:

- The console.
- A file.
- A set of files.

A different handler manages each output destination. For trace servers, the possible handlers are:

- com.ibm.logging.ConsoleHandler

- `com.ibm.logging.FileHandler` (a single file)
- `com.ibm.logging.MultiFileHandler` (a backed-up circular file)

Do not register a socket or ENA handler for a trace server; trace servers are used to route messages from trace clients already using socket and ENA handlers. The trace servers cannot also use them.

You register a handler by using the `epictracehandler` element. The `TraceServer` definition sends trace messages to the multifile handler. Other trace servers must specify one or more handlers; see Figure 6 on page 7 for information on registering more than one handler. Each registered handler must also be configured; see “Configuring handlers” for more information.

```
<ePICApplication epicappid="TraceServer">
  <AdapterRouting cn="epicadapterrouting">
    ...
  </AdapterRouting>
  <ePICTraceExtensions cn="epicappextensions">
    ...
    <epictracehandler>com.ibm.logging.MultiFileHandler</epictracehandler>
    ...
  </ePICTraceExtensions>
</ePICApplication>
```

Figure 14. The default `TraceServer` definition: handler registration

**Configuring handlers:** Each of the handlers has different requirements, so each needs its own configuration. For example, each of the handlers needs to know how to format its output. MQSeries Adapter Kernel provides the class `com.ibm.epic.trace.client.ReFormatter` for use by trace servers with console and file handlers. A file handler also needs to know to which files it writes. The `TraceServer` entry in the `aqmconfig.xml` file illustrates the configuration of both types of handler, but other trace servers need to configure only the handlers they register.

The `ePICTandler` element is used to configure individual handlers. The element uses the attribute `epictracehandler` to indicate the handler being configured; this allows multiple handlers to be configured. However, only handlers registered with the `epictracehandler` element are used.

*Multifile handler:* A trace server registering the multifile handler can indicate the formatter to be used by the handler, the base name for the files, the number of files, and the size of the files. MQSeries Adapter Kernel provides the class `com.ibm.epic.trace.client.ReFormatter` for use by trace servers with console and file handlers; this is the default formatter. The filename can be anything; the default base filename for the `TraceServer` is `trc.log`. The default number of files is three, and the default size 1,000,000 bytes (1000 one-kilobyte blocks).



The formatter can be explicitly specified within an `epictraceformatter` element; the file is specified within an `epictracefilename` element; the number of files, within an `epictracefilenumber` element; and the size (in one-kilobyte blocks), within an `epictracefilesize` element.

The multifile handler begins by writing a file called `trc1.log`, until that file reaches the specified size. It then copies `trc1.log` to the name `trc2.log` and begins overwriting the file `trc1.log`. When the file next reaches the specified size, the file `trc2.log` is copied to the name `trc3.log`, `trc1.log` is copied to `trc2.log`, and the handler begins overwriting `trc1.log` again. After the number of files reaches the specified value—in this case three, reached with `trc3.log`—the handler stops saving the oldest file, the one with the highest number. In this example, the file `trc3.log` is never copied to a new name. It is constantly replaced when the `trc2.log` file is copied.

```
<ePICAApplication epicappid="TraceServer">
  <AdapterRouting cn="epicadapterrouting">
    ...
  </AdapterRouting>
  <ePICTraceExtensions cn="epicappextensions">
    ...
    <epictracehandler>com.ibm.logging.MultiFileHandler</epictracehandler>
    <ePICTraceHandler epictracehandler="com.ibm.logging.MultiFileHandler">
      <epictraceformatter>
        com.ibm.epic.trace.client.ReFormatter
      </epictraceformatter>
      <epictracefilename>trc.log</epictracefilename>
      <epictracefilenumber>3</epictracefilenumber>
      <epictracefilesize>1000</epictracefilesize>
    </ePICTraceHandler>
    ...
  </ePICTraceExtensions>
</ePICAApplication>
```

Figure 15. Configuring a multifile handler in a trace server

*Console handler:* A trace server registering the console handler can also indicate the formatter to be used by the handler. MQSeries Adapter Kernel provides the class `com.ibm.epic.trace.client.ReFormatter` for use by trace servers with console and file handlers; this is the default. The formatter can be explicitly specified within an `epictraceformatter` element. Console handlers require no further configuration. Note that the trace server does not write to the console unless the console handler is registered in the `epictracehandler` element.

```

<ePICAApplication epicappid="TraceServer">
  <AdapterRouting cn="epicadapterrouting">
    ...
  </AdapterRouting>
  <ePICTraceExtensions cn="epicappextensions">
    ..
    <epictracehandler>com.ibm.logging.ConsoleHandler</epictracehandler>
    ...
    <ePICTraceHandler epictracehandler="com.ibm.logging.ConsoleHandler">
      <epictraceformatter>
        com.ibm.epic.trace.client.ReFormatter
      </epictraceformatter>
    </ePICTraceHandler>
  </ePICTraceExtensions>
</ePICAApplication>

```

Figure 16. Configuring a console handler for a trace server

### Configuring trace workers

Trace servers are used only when trace clients register socket or ENA handlers. The trace servers themselves make use of special trace clients called workers. Unlike regular trace clients, workers are not configured under separate application identifiers. The configuration of trace workers is embedded in the configuration of the trace server.

Because a trace server is configured for trace clients using socket or ENA handlers, the trace server's configuration contains an `ePICTraceHandler` element for either a socket handler or an ENA handler. However, these handlers cannot be registered by the trace server. Instead, the configuration for socket handlers is used by trace workers in a trace server receiving messages from a trace client with a socket handler registered. The ENA handler is similarly used by trace workers when the trace server receives messages from a trace client using an ENA handler.

The `TraceServer` entry in the `aqmconfig.xml` file illustrates the configuration of both type of workers, but other trace servers need to configure only the workers needed by trace clients.

**Socket Handlers:** Trace workers cannot be configured to use socket handlers, but if a trace client uses a socket handler to connect to a trace server and the default server host and port numbers are not being used, the trace server needs an entry for the socket handler. Specify the handler in the `ePICTraceHandler` element, and within that element, specify the host and port it connects to. The `epictracesocketserverhost` element indicates the host; it

defaults to "localhost." The `epictraceportnumber` element indicates the port; it defaults to 8181.

```
<ePICAApplication epicappid="TraceServer">
  <AdapterRouting cn="epicadapterrouting">
    ...
  </AdapterRouting>
  <ePICTraceExtensions cn="epicappextensions">
    ...
    <epictracehandler>com.ibm.logging.MultiFileHandler</epictracehandler>
    ...
    <ePICTraceHandler epictracehandler="com.ibm.logging.SocketHandler">
      <epictracesocketserverhost>localhost</epictracesocketserverhost>
      <epictraceportnumber>8181</epictraceportnumber>
    </ePICTraceHandler>
  </ePICTraceExtensions>
</ePICAApplication>
```

Figure 17. Configuring trace workers for socket handlers

**ENA Handlers:** Trace workers cannot be configured to use ENA handlers, but if the trace client uses an ENA handler to connect to a trace server, then the trace-server configuration must include an `AdapterRouting` element. This element is used by the client's ENA handler and by the trace-server daemon. See *Quick Beginnings* for more information on creating `AdapterRouting` elements.

```
<ePICAApplication epicappid="TraceServer">
  <AdapterRouting cn="epicadapterrouting">
    <epicmqppqueuemgr>DEFAULT</epicmqppqueuemgr>
    <ePICBodyCategory epicbodycategory="DEFAULT">
      <ePICBodyType epicbodytype="DEFAULT">
        <epicreceivemode>MQPP</epicreceivemode>
        <epicreceivemqppqueue>TraceServerAIQ</epicreceivemqppqueue>
      </ePICBodyType>
    </ePICBodyCategory>
  </AdapterRouting>
</ePICAApplication>
```

Figure 18. Configuring trace workers for ENA handlers

The complete configuration entry for the `TraceServer` application is shown in Figure 19 on page 18. This server registers only the multifile handler, but the entry also illustrates the configuration the console handler. That handler is not used unless it is registered instead of, or in addition to, the multifile handler. The server also configures a socket handler. This handler cannot be registered by the trace server; it is used by trace workers accepting messages from a trace client using a socket handler.

```

<ePICAApplication epicappid="TraceServer">
  <AdapterRouting cn="epicadapterrouting">
    <epicmqppqueuemgr>DEFAULT</epicmqppqueuemgr>
    <ePICBodyCategory epicbodycategory="DEFAULT">
      <ePICBodyType epicbodytype="DEFAULT">
        <epicreceivemode>MQPP</epicreceivemode>
        <epicreceivemqppqueue>TraceServerAIQ</epicreceivemqppqueue>
      </ePICBodyType>
    </ePICBodyCategory>
  </AdapterRouting>
  <ePICATraceExtensions cn="epicappextensions">
    <epictracesyncoperation>>false</epictracesyncoperation>
    <epictracemessagefile>
      com.ibm.epic.trace.server.TraceServerMessage
    </epictracemessagefile>
    <epictracehandler>com.ibm.logging.MultiFileHandler</epictracehandler>
    <ePICATraceHandler epictracehandler="com.ibm.logging.SocketHandler">
      <epictracesocketserverhost>localhost</epictracesocketserverhost>
      <epictraceportnumber>8181</epictraceportnumber>
    </ePICATraceHandler>
    <ePICATraceHandler epictracehandler="com.ibm.logging.ConsoleHandler">
      <epictraceformatter>
        com.ibm.epic.trace.client.ReFormatter
      </epictraceformatter>
    </ePICATraceHandler>
    <ePICATraceHandler epictracehandler="com.ibm.logging.MultiFileHandler">
      <epictraceformatter>
        com.ibm.epic.trace.client.ReFormatter
      </epictraceformatter>
      <epictracefilename>trc.log</epictracefilename>
      <epictracefilenumber>3</epictracefilenumber>
      <epictracefilesize>1000</epictracefilesize>
    </ePICATraceHandler>
  </ePICATraceExtensions>
</ePICAApplication>

```

*Figure 19. The complete default TraceServer definition*

---

## Reading trace messages

When tracing is requested, the kernel generates the trace messages and sends them to the desired destination. The contents of trace messages vary with the application and with the level of tracing requested, but they commonly include the following:

- Time stamp (date and time).
- Name of the class containing the traced method.
- Thread name (for example, main, Thread-1, Thread-2).
- Name and signature of the method traced.

- Organization (always "ePIC").
- Application identifier.
- Source logical identifier, if available.
- Destination logical identifier, if available.
- Body category, if available.
- Body type, if available.
- Message identifier, if available.
- Transaction identifier, if available.
- Trace level, if available.
- Trace information describing the event. This includes a message code—the string AQM followed by a four-digit number—and the corresponding information string.

The following example shows two typical trace messages:

```
2000.05.18 08:57:48.742 com.ibm.epic.adapters.eak.nativeadapter.EpicNativeAdapter
Thread Name=main getLMSInstanceForApplication(String, String, String) ePIC TEST2
TYPE_INFO AQM5003: <CommunicationMode> has a value <MQPP>
```

```
2000.05.18 09:02:27.883 com.ibm.epic.adapters.eak.nativeadapter.LMSMQ
Thread Name=Thread-2 getReceiveQName(String, String, String) ePIC TEST2
TYPE_INFO AQM5010: Results from AdapterDirectory for body type <DEFAULT>,
body category <DEFAULT> application id <TEST2>
attribute filter <epicreceivemppqueue> value <TEST2AIQ>
```

*Figure 20. Two trace messages*

The first message comes from the `getLMSInstanceForApplication` method in the `EpicNativeAdapter` class. The message was collected by the main thread for the application TEST2. The trace information is the string `<CommunicationMode> has a value of <MQPP>`.

The second message comes from the `getReceiveQName` method in the `LMSMQ` class. The message was collected by thread 2 for the application TEST2. The trace information is the string beginning `Results from Adapter Directory...`



---

## Chapter 2. Common configuration problems

This section describes some common configuration problems and potential solutions. Most of this information applies directly to the MQSeries Adapter Kernel, but it also addresses some MQSeries problems (see “Problems with MQSeries” on page 23), MQSeries Integrator problems (see “Problems with MQSeries Integrator” on page 24), and some miscellaneous problems (see “Miscellaneous problems” on page 26).

Many configuration problems can be solved with a single step: check your spelling. Misspelled queue names, class names, application identifiers, and so on can all cause errors that look much more severe than they are.

---

### Kernel configuration problems

#### File aqmsetup not found

**Problem:** The aqmsetup file was not found.

**Response:** Make sure the AQMSETUPFILE environment variable is set to the location of the aqmsetup file in the correct directory. Make sure the name of the file is correct.

#### File aqmconfig.xml not found

**Problem:** The aqmconfig.xml file was not found.

**Response:** Make sure that the aqmconfig.xml file exists in the correct directory.

**Response:** Locate the corresponding aqmsetup file and make sure the AQMCONFIG= entry points to the correct directory using a fully qualified path.

#### No response to messages

**Problem:** Messages are successfully received by the target adapter but no replies are received.

**Response:** Make sure that the acknowledgement flag is set in the header of the original message.

#### Target adapter not found

**Problem:** The target adapter was not found and loaded.

**Response:** Make sure that the target adapter specified in the exception message exists.

**Response:** Make sure that the CLASSPATH environment variable includes the bin directory.

## Error parsing body data

**Problem:** The target adapter cannot parse the body data of the message.

**Response:** If the body data is XML and refers to a DTD, the most likely cause is that the target adapter cannot locate the XML DTD. Make sure that the aqmsetup file contains the necessary information.

## Exceptions from native adapters

**Problem:** A native adapter (sending or receiving) throws an exception.

**Response:** Make sure the proper queue name is specified. If the native adapter belongs to a trace client, make sure that the queue name is specified for the trace server.

**Response:** Make sure a valid communications mode is specified.

**Response:** If the communications mode requires use of MQSeries, make sure the CLASSPATH environment variable includes the MQSeries JAR files.

## Adapter daemon shuts down

**Problem:** The adapter daemon shuts down.

**Response:** Check for the existence of an `EpicSystemExceptionFilesequence-number.log` file.

**Response:** Make sure sufficient memory is available. A Java `OutOfMemoryError` will cause an adapter worker to shut down. (See “Java memory problem: `OutOfMemoryError`” on page 27 for more information.)

**Response:** Determine if messages have been re-routed to an error queue, causing it to fill up. If so, determine why messages are going to the error queue.

## Messages received but not sent to applications

**Problem:** Messages are successfully removed from the incoming queues but not forwarded to the receiving application.

**Response:** Make sure that the message is not being routed to an error queue. Symptoms of this problem include:

- An inability to locate the target adapter command.
- Errors returned by the target adapter command.
- Generation of an `EpicSystemExceptionFilesequence-number.log` file.



## Messages not removed from incoming queues

**Problem:** Messages are not successfully removed from the incoming queues.

**Response:** Make sure the adapter daemon is running. (It can be started with the `aqmstrad` command.)

**Response:** Check for the existence of an `EpicSystemExceptionFilesequence-number.log` file.

## Trace errors involving socket handlers

**Problem:** Attempts to use a socket handler to collect trace messages fail.

**Response:** Make sure the trace server is running. (It can be started with the `aqmstrtd` command.)

## Instantiation of trace client fails

**Problem:** Attempts to instantiate trace clients fail.

**Response:** The most likely cause is an error in the configuration of the application or the trace client it tries to instantiate. Look for an `EpicSystemExceptionFilesequence-number.log` file.

## Trace clients or servers loop

**Problem:** A trace client or server gets stuck in a loop.

**Response:** Make sure that the trace client or server itself is not configured to have tracing on.

**Response:** Make sure that the trace server is not configured to use socket handlers or native-adapter handlers.

---

## Problems with MQSeries

Applications using adapters rely on an underlying transport to move messages from one application to another. Problems with the transport affect the applications above it. This section describes some typical problems that arise with IBM MQSeries.

### Queue not found

**Problem:** The named queue does not exist.

**Response:** Make sure that the name is spelled properly and does exist.

**Response:** Use MQSeries utilities to create the queue.

### Channel down

**Problem:** The MQSeries channel is not available.

**Response:** Use MQSeries utilities to ensure that all necessary components are running.

### **Queue depth exceeded**

**Problem:** The depth of the MQSeries queue has been exceeded.

**Response:** Set the value of the MQSeries environment variable MAX\_QUEUE\_DEPTH appropriately.

**Response:** Use MQSeries or application-specific utilities to remove elements from the queue.

**Response:** Determine if another problem has caused the elements to remain on the queue.

### **Logs exceed available space**

**Problem:** MQSeries log files have exceeded the space available in the file system.

**Response:** Examine the logs and determine which to save. Move the necessary logs to secondary storage and remove the rest from the file system.

### **MQSeries JAR files not found**

**Problem:** The MQSeries JAR files are not found.

**Response:** Make sure the CLASSPATH environment variable includes the directories containing the MQSeries JAR files.

### **Unable to load message catalog**

**Problem:** Unable to load the MQJI message catalogue.

**Response:** Make sure that the CLASSPATH environment variable includes the directory containing the mqji\*.properties files.

---

## **Problems with MQSeries Integrator**

### **Message not understood**

**Problem:** MQSeries Integrator does not understand the messages.

**Response:** Make sure that the correct communication mode (for example, MQRFH2) is specified.

### **Receiving messages from MQSI**

**Problem:** Applications get errors when receiving messages from MQSeries Integrator.

**Response:** Make sure that the communication mode of the receiving application matches that used by MQSI to send the message.

---

## Problems with JMS

Applications using adapters rely on an underlying transport to move messages from one application to another. Problems with the transport affect the applications above it. This section describes some typical problems that arise with applications using JMS.

**Note:** IBM's JMS implementation is built over MQSeries. If you are using IBM's JMS implementation, make sure that MQSeries is behaving correctly. For more information on problems with MQSeries, see "Problems with MQSeries" on page 23.

### JNDI initialization fails

**Problem:** Initialization of JNDI fails.

**Response:** If you are using FSContext, ensure that the CLASSPATH environment variable includes a path to the fscontext.jar file.

**Response:** If you are using FSContext, ensure that the PROVIDER\_URL variable points to an existing directory.

### JMS communication mode fails

**Problem:** Attempts to use the JMS communication mode fail with java.lang.NoClassDefFoundError exceptions.

**Response:** If you have not explicitly installed the JMS base package, download it from the MQSeries SupportPac site. MQSeries Adapter Kernel does not provide the JMS package.

**Response:** IBM's MQSeries JMS implementation requires several JAR files to be added to the CLASSPATH environment variable. (See that product's documentation for details.) Make sure that all the necessary JAR files are on the CLASSPATH environment variable.

### Errors locating QueueConnectionFactory or Queue objects

**Problem:** Applications report errors in locating QueueConnectionFactory or Queue objects.

**Response:** Make sure that the QueueConnectionFactory object has been created and stored.

**Response:** If using FSContext, make sure that the aqmconfig.xml configuration file includes a contextelement for the root directory.

## Binding problem on HP-UX

**Problem:** Applications cannot resolve the `libmqjbd01.so` library file, causing bindings to fail.

**Response:** Make sure the latest MQSeries Adapter Offering CSD is installed. You may also need to install a JMS SupportPac.

## Code-page conversion problem on HP-UX

**Problem:** Applications experience problems converting the default code page.

**Response:** Temporarily change the HP-UX default code page, `en_US.roman8(1051)`, to `en_US.iso88591 (819)` and recreate the `QManagerobject`.

---

## Miscellaneous problems

### Display errors on UNIX

**Problem:** Attempts to run the installation program on UNIX result in reports that the display cannot be opened.

**Response:** Make sure that the `DISPLAY` environment variable is set appropriately. The required value takes the name of the host machine followed by the string `:0.0`. The value `grizelda:0.0` is valid for a machine named `grizelda`.

**Response:** Use the X Window security program, `xhost`, to turn off access control for the console, by running the command with the `+` (plus) argument: `xhost +`. This command must be run from the console; it cannot be run remotely.

### Core dump on Solaris

**Problem:** Using Java JDK1.2.2 and MQSeries to send or receive messages results in `*** panic: libthread loaded into green threads Abort (coredump)`.

**Response:** Set the environment variable `THREADS_FLAG` to `"native"`; for example, `export THREADS_FLAG=native`.

### Segmentation violation on AIX

**Problem:** Running C-language adapters on AIX results in segmentation violation (`SIGSEGV`) and core dump.

**Response:** Set the environment variable `AIXTHREAD_SCOPE` to `"S"`; for example, `export AIXTHREAD_SCOPE=S`.

## Java memory problem: OutOfMemoryError

**Problem:** Applications generate OutOfMemoryErrors from Java, which can signal that applications are processing very large messages or that the adapters are using large amounts of memory.

**Response:** If the error originates from within a Java-initiated process, try increasing the memory values for Java at startup.

**Response:** If the error originates from within a C-language-initiated process, try increasing Java's memory values by setting the parameters in the aqmsetup file.

## XML parser problems

**Problem:** Errors parsing XML data.

**Response:** Make sure that the CLASSPATH environment variable points to the correct version of the XML4J parser, version 2\_0\_15.

## Using WebSphere

**Problem:** Environmental problems when also using WebSphere Application Server.

**Response:** Environment variables like CLASSPATH, PATH, and AQMSETUPFILE must be set within WebSphere's configuration tools.



---

## Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**  
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make

improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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 United Kingdom Laboratories,  
Mail Point 151,  
Hursley Park,  
Winchester,  
Hampshire,  
England  
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.



---

## Trademarks

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

AIX	OS/400
AS/400	RISC System/6000
IBM	RS/6000
MQSeries	WebSphere

Lotus and LotusScript are trademarks of Lotus Development Corporation in the United States, or other countries, or both.

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

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

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



---

# Index

## A

- aqmsetup file
  - aqmconfig.xml reference 21
  - environment variable 21
- AQMSETUPFILE
  - environment variable 21, 27

## C

- configuration
  - tracing 2, 4
- configuration problems
  - acknowledgement 21
  - adapter daemon 22, 23
  - AIX 26
  - aqmconfig.xml file 21
  - aqmsetup file 21
  - finding target adapter 21
  - Java memory 27
  - message reception 22, 23
  - native adapters 22
  - parsing body data 22
  - receiving messages 22
  - sending messages 22
  - socket handler 23
  - Solaris 26
  - target adapter 21, 22
  - trace client 22, 23
  - tracing 22, 23

- configuration problems (*continued*)
  - UNIX display 26
  - XML parser 27
- Configuration problems
  - WebSphere 27

## E

- environment variables
  - AQMSETUPFILE 21, 27
  - CLASSPATH 21, 27
  - PATH 27

## J

- JMS problems
  - binding on HP-UX 26
  - code-page conversion on HP-UX 26
  - communication mode fails 25
  - Finding Queue objects 25
  - Finding QueueConnectionFactory objects 25
  - JNDI initialization fails 25

## M

- MQSeries Integrator problems
  - message not understood 24
  - message reception 24

- MQSeries problems
  - channel down 23
  - JARs not found 24
  - logs exceed space 24
  - MQJI properties 24
  - queue depth exceeded 24
  - queue not found 23
  - unable to load message catalog 24

## T

- target adapter
  - configuration problems 21, 22
- trace components
  - handlers 1
  - trace client 1
  - trace server 1
  - trace worker 1
- tracing
  - application identifier 3
  - components 1
  - configuration 2, 4
  - console handler 2
  - correlating messages 2
  - ENA handler 2
  - file handler 2
  - handlers 2
  - native-adapter handler 2
  - output destinations 2
  - overview 1
  - socket handler 2
  - trace levels 3