

IBM WebSphere Application Server, Version 5



# Servers

**Note**

Before using this information, be sure to read the general information under “Trademarks and service marks” on page v.

**Compilation date: November 13, 2002**

**© Copyright International Business Machines Corporation 2002. All rights reserved.**

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

---

# Contents

**Trademarks and service marks . . . . . v**

**Chapter 1. Welcome to Servers. . . . . 1**

**Chapter 2. Configuring application servers . . . . . 3**

Application servers . . . . . 3

Creating application servers . . . . . 4

    Configuring application servers for UTF-8 encoding . . . . . 5

Managing application servers. . . . . 5

    Application server collection . . . . . 6

    Starting servers . . . . . 10

    Running servers as non-root using the console . 10

    Detecting and handling problems with run-time components . . . . . 11

    Stopping servers. . . . . 11

Transports . . . . . 11

Configuring transports . . . . . 12

    HTTP transport collection . . . . . 13

    HTTP transport settings . . . . . 13

    Example: Manually editing transport settings in the server.xml file . . . . . 14

Custom services . . . . . 15

Developing custom services . . . . . 15

    Custom service collection. . . . . 17

Process definition . . . . . 18

Defining application server processes. . . . . 18

    Process definition settings . . . . . 19

Java virtual machines (JVMs) . . . . . 21

Using the JVM . . . . . 21

Java virtual machine settings . . . . . 22

    Example: Configuring JVM sendRedirect calls to use context root . . . . . 24

Preparing to host applications . . . . . 25

Application servers: Resources for learning. . . . . 25

**Chapter 3. Managing Object Request Brokers . . . . . 27**

Object Request Brokers . . . . . 27

Object Request Broker tuning guidelines. . . . . 28

Object Request Broker service settings in administrative console. . . . . 28

    Request timeout . . . . . 29

    Request retries count . . . . . 29

    Request retries delay . . . . . 29

    Connection cache maximum. . . . . 29

    Connection cache minimum . . . . . 29

    ORB tracing . . . . . 30

    Locate request timeout . . . . . 30

    Force tunneling . . . . . 30

    Tunnel agent URL . . . . . 30

    Pass by reference . . . . . 31

Object Request Broker service settings that can be added to the administrative console . . . . . 31

Object Request Broker communications trace . . . 32

Client-side programming tips for the Java Object Request Broker service. . . . . 36

Character codeset conversion support for the Java

Object Request Broker service . . . . . 38

Object Request Brokers: Resources for learning . . 39



---

## Trademarks and service marks

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

- Everyplace
- iSeries
- IBM
- Redbooks
- ViaVoice
- WebSphere
- zSeries

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

Microsoft, Windows, Windows NT, and the Windows logo 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.

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



---

## Chapter 1. Welcome to Servers

The product provides application servers and more.

### **Application servers**

Application servers extend the ability of a Web server to handle Web application requests. An application server enables a server to generate a dynamic, customized response to a client request.

You can (configure one or more application servers) and enhance the operation of an application server using:

- (transports)
- (custom services)
- (command-line information) that passes to a server when it starts or initializes
- Settings that (improve the use of the Java virtual machine (JVM))

Application servers use an Object Request Broker (ORB) for RMI/IIOP communication.

### **Java Messaging (JMS) servers**

The product supports asynchronous messaging based on the Java Messaging Service (JMS) of a JMS provider that conforms to the JMS specification version 1.0.2 and supports the Application Server Facility (ASF) function defined within that specification.

For IBM WebSphere Application Server, the JMS functions (of the JMS provider) for an application server are served by the JMS server within the application server. For Network Deployment and Enterprise Extensions, the JMS functions (of JMS providers) within the administration domain are served by one or more JMS servers. There can be at most one JMS server on each node in the administration domain, and any application server within the domain can access JMS resources served by any JMS server on any node in the domain.

### **Web services servers**

The Web services components included with this product version build upon the Apache Simple Object Access Protocol (SOAP) 2.3-based capabilities delivered with Version 4.0.x.

New in Network Deployment are the following features:

- A private Universal Description, Discovery and Integration (UDDI) Registry, implementing Version 2.0 of the UDDI specification
- A Web Services Gateway for providing gateway access to existing Web services





---

## Chapter 2. Configuring application servers

An application server configuration provides settings that control how an application server provides services for running enterprise applications and their components.

This section describes how to create and configure application servers, and how to otherwise handle server configurations.

A WebSphere Application Server administrator can configure one or more application servers and perform tasks such as the following:

### Steps for this task

1. Create application servers.
2. Manage application servers.
3. **(Optional)** Configure transports.
4. **(Optional)** Develop custom services.
5. **(Optional)** Define processes for the application server. As part of defining processes, you can define process execution statements for starting or initializing a UNIX process, monitoring policies to track the performance of a process, process logs to which standard out and standard error streams write, and name-value pairs for properties.
6. **(Optional)** Use the Java virtual machine.

After preparing a server, deploy an application or component on the server. See "Preparing to host applications" for a sample procedure that you might follow in configuring the application server run-time and resources.

---

### Application servers

Application servers extend a Web server's capabilities to handle Web application requests, typically using Java technology. An application server makes it possible for a server to generate a dynamic, customized response to a client request.

For example, suppose—

1. A user at a Web browser on the public Internet visits a company Web site. The user requests to use an application that provides access to data in a database.
2. The user request flows to the Web server.
3. The Web server determines that the request involves an application containing resources not handled directly by the Web server (such as servlets). It forwards the request to a WebSphere Application Server product.
4. The WebSphere Application Server product forwards the request to one of its application servers on which the application is running.
5. The invoked application then processes the user request. For example:
  - An application servlet prepares the user request for processing by an enterprise bean that performs the database access.
  - The application produces a dynamic Web page containing the results of the user query.

6. The application server collaborates with the Web server to return the results to the user at the Web browser.

The WebSphere Application Server product provides multiple application servers that can be either separately configured processes or nearly identical clones.

---

## Creating application servers

You can create a new application server using the wsadmin tool or the Create New Application Server page of the administrative console. The steps below describe how to use the Create New Application Server page.

### Steps for this task

1. Go to the Application Servers page and click **New**. This brings you to the Create New Application Server page.
2. Follow the instructions on the Create New Application Server page and define your application server.
  - a. Select a node for the application server.
  - b. Type in a name for the application server. The name must be unique within the node.
  - c. Select whether the new server will have unique ports for each HTTP transport. By default, this option is enabled. If you select this option, you might need to update the alias list for the virtual host that you plan to use with this server to contain these new port values. If you deselect this option, ensure that the default port values do not conflict with other servers on the same physical machine.
  - d. Select a template to be used in creating the new server. You can use a default application server template for your new server or use an existing application server as a template. The new application server will inherit all properties of the template server.
  - e. If you create the new server using an existing application server as a model, select whether to map applications from the existing server to the new server. By default, this option is disabled.
3. **(Optional)** To use multiple language encoding support in the administrative console, configure an application server with UTF-8 encoding enabled.

### Results

The new application server appears in the list of servers on the Application Servers page.

### What to do next

Note that the application server created has many default values specified for it. An application server has many properties that can be set and creating an application server on the Create New Application Server page specifies values for only a few of the important properties. To view all of the properties of your application server and to customize your application server further, click on the name of your application server on the Application Servers page and change the settings for your application server as needed.

## Configuring application servers for UTF-8 encoding

To use multiple language encoding support in the administrative console, you must configure an application server with UTF-8 encoding enabled.

### Steps for this task

1. Create an application server or use an existing application server.
2. On the Application Server page, click on the name of the server you want enabled for UTF-8.
3. On the settings page for the selected application server, click **Process Definition**.
4. On the Process Definition page, click **Java Virtual Machine**.
5. On the Java Virtual Machine page, specify `-Dclient.encoding.override=UTF-8` for **Generic JVM Arguments** and click **OK**.
6. Click **Save** on the console taskbar.
7. Restart the application server.

Note that the `autoRequestEncoding` option does not work with UTF-8 encoding enabled. The default behavior for WebSphere Application Server is, first, to check if `charset` is set on content type header. If it is, then the product uses content type header for character encoding; if it is not, then the product uses character encoding set on server using the system property `default.client.encoding`. If `charset` is not present and the system property is not set, then the product uses ISO-8859-1. Enabling `autoRequestEncoding` on a Web module changes the default behavior: if `charset` is not present on an incoming request header, the product checks the `Accept-Language` header of the incoming request and does encoding using the first language found in that header. If there is no `charset` on content type header and no `Accept language` header, then the product uses character encoding set on server using the system property `default.client.encoding`. As with the default behavior, if `charset` is not present and the system property is not set, then the product uses ISO-8859-1.

---

## Managing application servers

To view information about an application server, use the Application Servers page. For the Network Deployment product, you can also use the Application Servers page to manage application servers. For the single-server (base) product, you cannot manage application servers from the administrative console; you must manage application servers from a console hosted by a Network Deployment deployment manager, use the `wasadmin` tool, or use command line tools such as `startServer` and `stopServer`.

### Steps for this task

1. Access the Application Servers page. Click **Servers > Application Servers** in the console navigation tree.
2. View information about application servers. The Application Servers page lists application servers in the cell and the nodes holding the application servers.

To view additional information about a particular application server or to further configure a server, click on the server's name under **Name**. This accesses the settings page for an application server.

To view product information for a server:

- a. Ensure that the server is running.
- b. Go to the **Runtime** tab on the settings page for an application server.

c. Click **Product Information**.

The Product Information page displayed lists the WebSphere Application Server products installed for the server, the version and build levels for the products, the build dates, and any e-fixes applied to the server.

3. Create an application server. Click **New** and follow the instructions on the Create New Application Server page.
4. Monitor the running of application servers.
5. **(Optional)** Delete an application server.
  - a. Click **Servers > Application Servers** in the console navigation tree to access the Application Servers page.
  - b. Place a checkmark in the check box beside the server you want deleted.
  - c. Click **Delete**.
  - d. Click **OK** to confirm the deletion.

## Application server collection

Use this page to view information about and manage application servers.

The Application Servers page lists application servers in the cell and the nodes holding the application servers.

To view this administrative console page, click **Servers > Application Servers**.

### Name

Specifies a logical name for the server. Server names must be unique within a node.

### Node

Specifies the name of the node for the application server.

### Status

Indicates whether the application server is started or stopped.

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.

## Application server settings

Use this page to view or change the settings of an application server instance.

To view this administrative console page, click **Servers > Application Servers > server\_name**.

The **Configuration** tab provides editable fields and the **Runtime** tab provides read-only information. The **Runtime** tab is available only when the server is running.

**Name:** Specifies a logical name for the server. Server names must be unique within a node.

Data type	String
Default	server1

**Initial State:** Specifies the desired state of the application server when its containing process (node) starts. The options are *Started* and *Stopped*. The default is *Started*.

Data type	String
Default	Started

**Application Classloader Policy:** Specifies whether to use a single classloader to load all applications or to use a different classloader for each application.

The options are SINGLE and MULTIPLE. The default is to use a separate classloader for each application (MULTIPLE).

Data type	String
Default	MULTIPLE

**Application Classloading Mode:** Specifies whether the classloader should search in the parent classloader or in the application classloader first to load a class. The standard for JDK classloaders and WebSphere classloaders is PARENT\_FIRST. By specifying PARENT\_LAST, your application can override classes contained in the parent classloader, but this action can potentially result in ClassCastException or LinkageErrors if you have mixed use of overridden classes and non-overridden classes.

The options are PARENT\_FIRST and PARENT\_LAST. The default is to search in the parent classloader before searching in the application classloader to load a class.

Data type	String
Default	PARENT_FIRST

**Process ID:** Specifies a string identifying the process.

Data type	String
-----------	--------

**Cell Name:** Specifies the name of the cell for the application server.

Data type	String
Default	<i>host_name</i> Network

**Node Name:** Specifies the name of the node for the application server.

Data type	String
-----------	--------

**State:** Indicates whether the application server is started or stopped.

Data type	String
Default	Started

**End point collection:** Use this page to view and manage communication end points used by run-time components running within a process. End points provide host and port specifications for a server.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > End Points**.

Note that this page displays only when you are working with end points for application servers.

**End Point Name:** Specifies the name of an end point. Each name must be unique within the server.

**End point settings:** Use this to view and change the configuration for a communication end point used by run-time components running within a process. An end point provides host and port specifications for a server.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > End Points > *end\_point\_name***.

**End Point Name:** Specifies the name of the end point. The name must be unique within the server.

Note that this field displays only when you are defining an end point for an application server.

Data type	String
-----------	--------

**Host:** Specifies the IP address, domain name server (DNS) host name with domain name suffix, or just the DNS host name, used by a client to request a resource (such as the naming service, administrative service, or JMS broker).

For example, if the host name is myhost, the fully qualified DNS name can be myhost.myco.com and the IP address can be 155.123.88.201.

Data type	String
Default	*

**Port:** Specifies the port for which the service is configured to accept client requests. The port value is used in conjunction with the host name.

Data type	Integer
Default	None

**Property collection:** Use this page to view and manage arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > Custom Properties**.

**Name:** Specifies the name (or key) for the property.

**Value:** Specifies the value paired with the specified name.

**Description:** Provides information about the name-value pair.

**Property settings:** Use this page to configure arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties. Defining a new property enables you to configure a setting beyond that which is available in the administrative console.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > Custom Properties > *property\_name***.

**Name:** Specifies the name (or key) for the property.

Data type	String
-----------	--------

**Value:** Specifies the value paired with the specified name.

Data type	String
-----------	--------

**Description:** Provides information about the name-value pair.

Data type	String
-----------	--------

**Server component collection:** Use this page to view information about and manage server component types such as application servers, messaging servers, or name servers.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > Server Components**.

**Type:** Specifies the type of internal server.

**Server component settings:** Use this page to view or configure a server component instance.

To view this administrative console, click **Servers > Application Servers > *server\_name* > Server Components > *server\_component\_name***.

**Name:** Specifies the name of the component.

Data type	String
-----------	--------

**Initial State:** Specifies the desired state of the component when the server process starts. The options are: *Started* and *Stopped*. The default is *Started*.

Data type	String
Default	Started

**Thread pool settings:** Use this page to configure a group of threads that an application server uses. A thread pool enables components of the server to reuse threads to eliminate the need to create new threads at run time. Creating new threads expends time and resources.

To view this administrative console page, click **Servers > Manage Application Servers > *server\_name* > ORB Service > Thread Pool**. (You can reach this page through more than one navigational route.)

**Minimum size:** Specifies the minimum number of threads to allow in the pool.

Data type	Integer
Default	10

**Maximum size:** Specifies the maximum number of threads to allow in the pool.

Data type	Integer
Default	50

**Thread inactivity timeout:** Specifies the number of milliseconds of inactivity that should elapse before a thread is reclaimed. A value of 0 indicates not to wait and a negative value (less than 0) means to wait forever.

Data type	Integer
Units	Milliseconds
Default	3500

**Growable thread pool:** Specifies whether the number of threads can increase beyond the maximum size configured for the thread pool.

Data type	Boolean
Default	Not enabled (false)
Range	Valid values are <b>Allow thread allocation beyond maximum thread size</b> or Not enabled.

## Starting servers

Starting a server starts a new server process based on the current server configuration's process definition settings.

### Starting a server from a command line

To start a server, run the startServer command.

### Starting a server for tracing and debugging

To start the server with standard Java debugging enabled:

1. Click **Servers > Application Servers** from the administrative console navigation tree. Then, click the application server whose processes you want to trace and debug, **Process Definition**, and **Java Virtual Machine**.
2. On the Java Virtual Machine page, place a checkmark in the check box for the **Debug Mode** setting to enable the standard Java debugger. If needed, set debug arguments. Then, click **OK**.
3. Save the changes to a configuration file.
4. Stop the server.
5. Start the server again (see above).

## Running servers as non-root using the console

By default, WebSphere Application Server servers use a root ID. A server can be run using a non-root ID if security file system permissions grant all users of a certain group writable access to main WebSphere Application Server directories and the user ID and group ID for the server "run as" the user and group.

### Steps for this task

1. Specify user and group ID values for the **Run As User** and **Run As Group** settings for a server



- a. Go to the Process Execution page for the server you want to run as non-root. Click **Servers > Application Servers > *server\_name* > Process Definition > Process Execution**.
- b. For **Run As User**, specify a user name for the process to run as.
- c. For **Run As Group**, specify a group name for the process to run as.
2. Using operating system tools, create a set of users that are all in the group.
3. Using operating system tools, change the permissions of the WebSphere Application Server installation root (*install\_root*) directory.
  - a. Change the group owner to the group.
  - b. Make the following files under the *install\_root* directory writable by the group:
    - Log files
    - All files and subdirectories below the tranlog directory
    - All files and subdirectories below the config/temp directory
4. From the user ID, run the startServer command to start the server.

## Detecting and handling problems with run-time components

You must monitor the status of run-time components to ensure that, once started, they remain operational as needed.

### Steps for this task

1. Regularly examine the status of run-time components.
 

One way is using the Logging and Tracing page of the administrative console. Click **Troubleshooting > Logs and Trace** in the console navigation tree to access the page.

Another way is to browse messages displayed under **WebSphere Runtime Messages** in the WebSphere status area at the bottom of the console. The run-time event messages marked with a red X provide detailed information on event processing.
2. If an application stops running when it should be operational, examine the application's status on an Applications page and try restarting the application.
3. If the run-time components do not restart, re-examine the messages and read information on problem determination to help you to restart the components.

## Stopping servers

Stopping a server stops a server process based on the current server configuration's process definition settings.

### Stopping a server from a command line

To stop a server, run the stopServer command.

---

## Transports

A transport is the request queue between a WebSphere Application Server plug-in for Web servers and a Web container in which the Web modules of an application reside. When a user at a Web browser requests an application, the request is passed to the Web server, then along the transport to the Web container.

Transports define the characteristics of the connections between a Web server and an application server, across which requests for applications are routed. Specifically, they define the connection between the Web server plug-in and the Web container of the application server.

Administering transports is closely related to administering WebSphere Application Server plug-ins for Web servers. Indeed, without a plug-in configuration, a transport configuration is of little use.

### The internal transport

For applications in a test or development environment (in other words, a non-production environment), you can use the internal HTTP transport system to serve servlets without an Web server plug-in. Simply use the internal HTTP transport port (typically on port 9080).

For example, to serve a servlet (*servlet\_path\_name*) without an HTTP server, use the URL: `http://server_name:port/servlet_path_name`

with *port* being the internal transport port number (typically 9080) and *server\_name* being `localhost` if the application server is on the local machine.

For a production environment, do not use the internal transport, as it lacks the performance available when using a Web server plug-in.

At times, you might be able to configure the internal transport to use a port other than 9080. The transport configuration is a part of the Web container configuration. To change the port number, you must adjust your virtual host alias and what you type into the Web browser.

---

## Configuring transports

You configure transports to specify:

- How to manage a set of connections. For example, to specify the number of concurrent requests to allow.
- Whether to secure the connections with SSL
- Host and IP information for the transport participants

### Steps for this task

1. Create an HTTP transport.
  - a. Ensure that virtual host aliases include port values for the new transport.
  - b. Go to the HTTP Transports page and click **New**.
  - c. On the settings page for an HTTP transport, specify values such as the transport's host name and port number, then click **OK**.
2. **(Optional)** Change the configuration for an existing transport.
  - a. Ensure that virtual host aliases include port values for the transport your are changing.
  - b. Go to the HTTP Transports page and click on the transport under **Host** whose configuration you want to change.
  - c. On the settings page for an HTTP transport, which might have the page title `DefaultSSLSettings`, change the specified values as needed, then click **OK**.
3. Regenerate the WebSphere plug-in for the Web server.

If the Web server is located on a machine remote from the application server, you might also need to perform special configuration tasks to redirect application requests from the Web server machine to the application server machine.

## HTTP transport collection

Use this page to view or manage HTTP transports. Transports provide request queues between WebSphere plug-ins for Web servers and Web containers in which the Web modules of applications reside. When you request an application in a Web browser, the request is passed to the Web server, then along the transport to the Web container.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > Web Container > HTTP Transports**.

### Host

Specifies the host IP address to bind for transport. If the application server is on a local machine, the host name might be localhost.

### Port

Specifies the port to bind for transport. The port number can be any port that currently is not in use on the system. The port number must be unique for each application server instance on a given machine.

### SSL Enabled

Specifies whether to protect connections between the WebSphere plug-in and application server with Secure Sockets Layer (SSL). The default is not to use SSL.

## HTTP transport settings

Use this page to view and configure an HTTP transport. The name of the page might be that of an SSL setting such as DefaultSSLSettings.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > Web Container > HTTP Transports > *host\_name***.

### Host

Specifies the host IP address to bind for transport.

If the application server is on a local machine, the host name might be localhost.

Data type	String
-----------	--------

### Port

Specifies the port to bind for transport. The port number can be any port that currently is not in use on the system. The port number must be unique for each application server instance on a given machine.

Data type	Integer
-----------	---------

### SSL Enabled

Specifies whether to protect connections between the WebSphere plug-in and application server with Secure Sockets Layer (SSL). The default is not to use SSL.

Data type	Boolean
Default	false

## SSL

Specifies the Secure Sockets Layer (SSL) settings type for connections between the WebSphere plug-in and application server. The options include one or more SSL settings defined in the Security Center; for example, DefaultSSLSettings, ORBSSLSettings, or LDAPSSLSettings.

Data type	String
Default	An SSL setting defined in the Security Center

## Example: Manually editing transport settings in the server.xml file

WebSphere Application Server Version 4.x has several transport properties that are not shown in the settings page for an HTTP transport:

### ConnectionIOTimeout

Specifies the maximum number of seconds to wait when trying to read or write data during a request.

### ConnectionKeepAliveTimeout

Specifies the maximum number of seconds to wait for the next request on a keep alive connection.

### MaxKeepAliveConnections

Specifies the maximum number of concurrent keep alive (persistent) connections across all HTTP transports. The default value is 90% of the maximum number of threads in the Web container thread pool. This prevents all of the threads from being held by keep alive connections so that there are threads available to handle new incoming connect requests.

### MaxKeepAliveRequests

Specifies the maximum number of requests which can be processed on a single keep alive connection.

To specify values for these transport properties, you can edit the *<properties>* settings in the server.xml file shown in **bold** font below. After editing the server.xml file, restart the server and regenerate the Web server plug-in. As to each of the new properties, whatever value you specify for a first transport is applied globally to all other HTTP transports in the cell.

```
<components xmi:type="applicationserver.webcontainer:WebContainer"
  xmi:id="WebContainer_1" enableServletCaching="false">
  <stateManagement xmi:id="StateManageable_5" initialState="START"/>
  <properties xmi:id="WebContainer_Property_1" name="MaxConnectBacklog"
    value="50"/>
  <properties xmi:id="WebContainer_Property_2"
    name="MaxKeepAliveConnections" value="45"/>
  <properties xmi:id="WebContainer_Property_3"
    name="MaxKeepAliveRequests" value="100"/>
  <properties xmi:id="WebContainer_Property_4"
    name="ConnectionIOTimeout" value="5"/>
  <properties xmi:id="WebContainer_Property_5"
    name="ConnectionKeepAliveTimeout" value="5"/>
<services xmi:type="applicationserver.webcontainer:SessionManager"
  xmi:id="SessionManager_1" enable="true" enableUrlRewriting="false"
  enableCookies="true" enableSSLTracking="false"
  enableProtocolSwitchRewriting="false"
  enableSecurityIntegration="false"
  sessionPersistenceMode="NONE" allowSerializedSessionAccess="false"
  accessSessionOnTimeout="true" maxWaitTime="5">
  <defaultCookieSettings xmi:id="Cookie_1" name="JSESSIONID" domain=""
    maximumAge="-1" path="/" secure="false"/>
```

```

<sessionDatabasePersistence xmi:id="SessionDatabasePersistence_1"
    datasourceJNDIName="jdbc/Sessions" userId="db2admin"
    password="db2admin" db2RowSize="ROW_SIZE_4KB"
    tableSpaceName=""/>
<tuningParams xmi:id="TuningParams_1" usingMultiRowSchema="false"
    maxInMemorySessionCount="1000" allowOverflow="true"
    invalidationTimeout="30" writeContents="ONLY_UPDATED_ATTRIBUTES"
    writeFrequency="TIME_BASED_WRITE" writeInterval="120"
    scheduleInvalidation="false">
    <invalidationSchedule xmi:id="InvalidationSchedule_1" firstHour="14"
        secondHour="2"/>
</tuningParams>
</services>
<threadPool xmi:id="ThreadPool_2" minimumSize="10" maximumSize="50"
    inactivityTimeout="3500" isGrowable="false"/>
<transports xmi:type="applicationserver.webcontainer:HTTPTransport"
    xmi:id="HTTPTransport_1" sslEnabled="false">
    <address xmi:id="EndPoint_1" host="" port="9080"/>
</transports>
<transports xmi:type="applicationserver.webcontainer:HTTPTransport"
    xmi:id="HTTPTransport_2" sslEnabled="true"
    sslConfig="DefaultSSLSettings">
    <address xmi:id="EndPoint_2" host="" port="9443"/>
</transports>
<transports xmi:type="applicationserver.webcontainer:HTTPTransport"
    xmi:id="HTTPTransport_3" sslEnabled="false">
    <address xmi:id="EndPoint_3" host="" port="9090"/>
</transports>
<transports xmi:type="applicationserver.webcontainer:HTTPTransport"
    xmi:id="HTTPTransport_4" sslEnabled="true"
    sslConfig="DefaultSSLSettings">
    <address xmi:id="EndPoint_4" host="" port="9043"/>
</transports>
</components>

```

---

## Custom services

A custom service provides the ability to plug into a WebSphere application server to define a hook point that runs when the server starts and shuts down.

A developer implements a custom service containing a class that implements a particular interface. The administrator configures the custom service in the administrative console, identifying the class created by the developer. When an application server starts, any custom services defined for the application server are loaded and the server run-time calls their initialize methods.

---

## Developing custom services

To define a hook point to be run when a server starts and shuts down, you develop a custom service class and then use the administrative console to configure a custom service instance for an application server. When the application server starts, the custom service starts and initializes.

### Steps for this task

1. Develop a custom service class that implements the ConfigService.html file described in Javadoc (`../javadoc/ae/com/ibm/websphere/management/configservice/ConfigService.html`).

The properties passed by the application server run-time to the initialize method can include one for an external file containing configuration information for the service (retrieved with `externalConfigURLKey`). In addition,

the properties can contain any name-value pairs that are stored for the service, along with the other system administration configuration data for the service. The properties are passed to the initialize method of the service as a Properties object.

There is a shutdown method for the interface as well. Both methods of the interface declare that they may throw an exception, although no specific exception subclass is defined. If an exception is thrown, the run-time logs it, disables the custom service, and proceeds with starting the server.

2. On the Custom Service page of the administrative console, click **New**. Then, on the settings page for a custom service instance, create a custom service configuration for an existing application server, supplying the name of the class implemented.

If your custom services class requires a configuration file, specify the fully-qualified path name to that configuration file in the **externalConfigURL** field. This file name is passed into your custom service class.

3. Stop the application server and then restart the server.
4. Check the application server to ensure that the initialize method of the custom service ran as intended. Also ensure that the shutdown method performs as intended when the server stops.

### Usage scenario

As mentioned above, your custom services class must implement the CustomService interface. In addition, your class must implement the initialize and shutdown methods. Suppose the name of the class that implements your custom service is *ServerInit*, your code would declare this class as shown below. The code below assumes that your custom services class needs a configuration file. It shows how to process the input parameter in order to get the configuration file. If your class does not require a configuration file, the code that processes configProperties is not needed.

```
public class ServerInit implements CustomService
{
/**
 * The initialize method is called by the application server run-time when the
 * server starts. The Properties object passed to this method must contain all
 * configuration information necessary for this service to initialize properly.
 *
 * @param configProperties java.util.Properties
 */
    static final java.lang.String externalConfigURLKey =
        "com.ibm.websphere.runtime.CustomService.externalConfigURLKey";

    static String ConfigFileName="";

    public void initialize(java.util.Properties configProperties) throws Exception
    {
        if (configProperties.getProperty(externalConfigURLKey) != null)
        {
            ConfigFileName = configProperties.getProperty(externalConfigURLKey);
        }

        // Implement rest of initialize method
    }

/**
 * The shutdown method is called by the application server run-time when the
 * server begins its shutdown processing.
 *
 * @param configProperties java.util.Properties
 */
}
```

```

public void shutdown() throws Exception
{
    // Implement shutdown method
}

```

## Custom service collection

Use this page to view a list of services available to the application server and to see whether the services are enabled. A custom service provides the ability to plug into a WebSphere application server and define code that runs when the server starts or shuts down.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > Custom Services**.

### External Configuration URL

Specifies the URL for a custom service configuration file.

If your custom services class requires a configuration file, the value provides a fully-qualified path name to that configuration file. This file name is passed into your custom service class.

### Classname

Specifies the class name of the service implementation. This class must implement the Custom Service interface.

### Display Name

Specifies the name of the service.

### Startup

Specifies whether the server attempts to start and initialize the service when its containing process (the server) starts. By default, the service is not enabled when its containing process starts.

### Custom service settings

Use this page to configure a service that runs in an application server.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > Custom Services > *custom\_service\_name***.

**Startup:** Specifies whether the server attempts to start and initialize the service when its containing process (the server) starts. By default, the service is not enabled when its containing process starts. To enable the service, place a checkmark in the check box.

Data type	Boolean
Default	false

**External Configuration URL:** Specifies the URL for a custom service configuration file.

If your custom services class requires a configuration file, specify the fully-qualified path name to that configuration file for the value. This file name is passed into your custom service class.

Data type	String
Units	URL

**Classname:** Specifies the class name of the service implementation. This class must implement the Custom Service interface.

Data type	String
Units	Java class name

**Display Name:** Specifies the name of the service.

Data type	String
-----------	--------

**Description:** Describes the custom service.

Data type	String
-----------	--------

**Classpath:** Specifies the class path used to locate the classes and JAR files for this service.

Data type	String
Units	Class path

---

## Process definition

A process definition specifies the run-time characteristics of an application server process.

A process definitions can include characteristics such as JVM settings, standard in, error and output paths, and the user ID and password under which a server runs.

---

## Defining application server processes

To enhance the operation of an application server, you can define command-line information for starting or initializing an application server process. Such settings define run-time properties such as the program to run, arguments to run the program, the working directory.

### Steps for this task

1. Go to the settings page for a process definition in the administrative console. Click **Servers > Application Servers** in the console navigation tree, click on an application server name and then **Process Definition**.
2. On the settings page for a process definition, specify the name of the executable to run, any arguments to pass when the process starts running, and the working directory in which the process will run. Then click **OK**.
3. **(Optional)** Specify process execution statements for starting or initializing a UNIX process.
4. **(Optional)** Specify monitoring policies to track the performance of a process.
5. **(Optional)** Specify process logs to which standard out and standard error streams write. Complete this step if you do not want to use the default file names.
6. **(Optional)** Specify name-value pairs for properties needed by the process definition.
7. Stop the application server and then restart the server.



8. Check the application server to ensure that the process definition runs and operates as intended.

## Process definition settings

Use this page to view or change settings for a process definition, which provides command-line information for starting or initializing a process.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > Process Definition**.

### Executable Name

Specifies the executable name of the process.

Data type String

### Executable Arguments

Specifies executable commands that run when the process starts.

For example, the executable target program might expect three arguments: *arg1 arg2 arg3*.

Data type String  
Units Java command-line arguments

### Working Directory

Specifies the file system directory in which the process will run.

This directory is used to determine the locations of input and output files with relative path names.

Passivated enterprise beans are placed in the current working directory of the application server on which the beans are running. Make sure the working directory is a known directory under the root directory of the WebSphere Application Server product.

Data type String

## Process execution settings

Use this page to view or change command-line information for starting or initializing a UNIX process.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > Process Definition > Process Execution**.

**Process Priority:** Specifies the operating system priority for the process. Only root users can change this value.

Data type Integer  
Default 1000 for WebSphere Application Server on most operating systems. On OS/400, the default is 25.

**UMASK:** Specifies the user mask under which the process runs (the file-mode permission mask).

Data type Integer

**Run As User:** Specifies the user that the process runs as.

Data type String

**Run As Group:** Specifies the group that the process is a member of and runs as.

On OS/400, the Run As Group setting is ignored.

Data type String

**Run In Process Group:** Specifies a specific process group for the process. This process group is useful for such things as processor partitioning. A system administrator can assign a process group to run on, for example, 6 of 12 processors. The default (0) is not to assign the process to any specific group.

On OS/400, the Run In Process Group setting is ignored.

Data type Integer

Default 0

## Process logs settings

Use this page to view or change settings for specifying the files to which standard out and standard error streams write.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > Process Definition > Process Logs**.

**Stdout File Name:** Specifies the file to which the standard output stream are directed. The file name can include a symbolic path name defined in the variable entries.

To direct server output to the administrative console or to the process that launched the server, either delete the value for this property or specify console.

Data type String

Units File path name

**Stderr File Name:** Specifies the file to which the standard error stream is directed. The file name can include a symbolic path name defined in the variable entries.

Data type String

Units File path name

## Monitoring policy settings

Use this page to view or change settings that control how the node agent monitors and restarts a process.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > Process Definition > Monitoring Policy**.

**Maximum Startup Attempts:** Specifies the maximum number of times to attempt to start the application server before giving up.

Data type Integer

**Ping Interval:** Specifies the frequency of communication attempts between the parent process, such as the node agent, and the process it has spawned, such as an application server. Adjust this value based on your requirements for restarting failed servers. Decreasing the value detects failures sooner; increasing the value reduces the frequency of pings, reducing system overhead.

Data type Integer

**Ping Timeout:** When a parent process is spawning a child process, such as when a process manager spawns a server, the parent process pings the child process to see whether the child was spawned successfully. This value specifies the number of seconds that the parent process should wait (after pinging the child process) before assuming that the child process failed.

Data type Integer  
Units Seconds

**Automatic Restart:** Specifies whether the process should restart automatically if it fails. The default is to restart the process automatically.

Data type Boolean  
Default true

**Node Restart State:** Specifies the desired state for the process after the node completely shuts down and restarts. The options are: *STOPPED*, *RUNNING*, *PREVIOUS*. The default is *STOPPED*.

Data type String  
Default STOPPED  
Range Valid values are STOPPED, RUNNING, or PREVIOUS.

---

## Java virtual machines (JVMs)

The Java virtual machine (JVM) is an interpretive computing engine responsible for executing the byte codes in a compiled Java program. The JVM translates the Java byte codes into the native instructions of the host machine. The application server, being a Java process, requires a JVM in order to run, and to support the Java applications running on it. JVM settings are part of an application server configuration.

---

## Using the JVM

As part of configuring an application server, you might define settings that enhance your system's use of the Java virtual machine (JVM).

To view and change the JVM configuration for an application server's process, use the Java Virtual Machine page of the console or use `wsadmin` to change the configuration through scripting.

### Steps for this task

1. Access the Java Virtual Machine page.
  - a. Click **Servers > Application Servers** in the console navigation tree.
  - b. On the Application Server page, click on the name of the server whose JVM settings you want to configure.
  - c. On the settings page for the selected application server, click **Process Definition**.
  - d. On the Process Definition page, click **Java Virtual Machine**.
2. On the Java Virtual Machine page, specify values for the JVM settings as needed and click **OK**.
3. Click **Save** on the console taskbar.
4. Restart the application server.

### Usage scenario

"Configuring application servers for UTF-8 encoding" provides an example that involves specifying a value for the **Generic JVM Arguments** property on the Java Virtual Machine page to enable UTF-8 encoding on an application server. Enabling UTF-8 allows multiple language encoding support to be used in the administrative console.

"Example: Configuring JVM sendRedirect calls to use context root" provides an example that involves defining a property for the JVM.

## Java virtual machine settings

Use this page to view and change the Java virtual machine (JVM) configuration for the application server's process.

To view this administrative console page, click **Servers > Application Servers > *server\_name* > Process Definition > Java Virtual Machine**.

### **Classpath**

Specifies the standard class path in which the Java virtual machine code looks for classes.

Enter each classpath entry into a table row. You do not need to add the colon or semicolon at the end of each entry.

Data type	String
Units	Class path

### **Boot Classpath**

Specifies bootstrap classes and resources for JVM code. This option is only available for JVM instructions that support bootstrap classes and resources. You can separate multiple paths by a colon (:) or semi-colon (;), depending on operating system of the node.

Data type	String
-----------	--------

### **Verbose Class Loading**

Specifies whether to use verbose debug output for class loading. The default is not to enable verbose class loading.

Data type	Boolean
Default	false

### **Verbose Garbage Collection**

Specifies whether to use verbose debug output for garbage collection. The default is not to enable verbose garbage collection.

Data type	Boolean
Default	false

### **Verbose JNI**

Specifies whether to use verbose debug output for native method invocation. The default is not to enable verbose Java Native Interface (JNI) activity.

Data type	Boolean
Default	false

### **Initial Heap Size**

Specifies the initial heap size available to the JVM code, in megabytes. The default is 64 for OS/400 and zero (0) for all other platforms.

Data type	Integer
Default	64 for OS/400, 0 for all other platforms

### **Maximum Heap Size**

Specifies the maximum heap size available to the JVM code, in megabytes. The default is zero (0) for OS/400 and 256 for all other platforms.

Data type	Integer
Default	0 for OS/400, 256 for all other platforms

### **Run HProf**

Specifies whether to use HProf profiler support. To use another profiler, specify the custom profiler settings using the HProf Arguments setting. The default is not to enable HProf profiler support.

If you set the Run HProf property to true, then you must specify command-line profiler arguments as values for the HProf Arguments property.

Data type	Boolean
Default	false

### **HProf Arguments**

Specifies command-line profiler arguments to pass to the JVM code that starts the application server process. You can specify arguments when HProf profiler support is enabled.

HProf arguments are only required if the Run HProf property is set to true.

Data type	String
-----------	--------

## Debug Mode

Specifies whether to run the JVM in debug mode. The default is not to enable debug mode support.

If you set the Debug Mode property to true, then you must specify command-line debug arguments as values for the Debug Arguments property.

Data type	Boolean
Default	false

## Debug Arguments

Specifies command-line debug arguments to pass to the JVM code that starts the application server process. You can specify arguments when Debug Mode is enabled.

Debug arguments are only required if the Debug Mode property is set to true.

Data type	String
Units	Java command-line arguments

## Generic JVM Arguments

Specifies command-line arguments to pass to the Java virtual machine code that starts the application server process.

Data type	String
Units	Java command-line arguments

## Executable JAR File Name

Specifies a full path name for an executable JAR file that the JVM code uses.

Data type	String
Units	Path name

## Disable JIT

Specifies whether to disable the Just In Time (JIT) compiler option of the JVM code. The default is not to disable JIT support.

Data type	Boolean
Default	false

## Operating System Name

Specifies JVM settings for a given operating system. When started, the process uses the JVM settings for the operating system of the node.

Data type	String
-----------	--------

## Example: Configuring JVM sendRedirect calls to use context root

If the `com.ibm.websphere.sendredirect.compatibility` property is not set and your application servlet code has statements such as `sendRedirect("/home.html")`, your Web browser might display messages such as *Error 404: No target servlet configured for uri: /home.html*. To instruct the server not to use the Web server's document root

and to use instead the Web application's context root for sendRedirect() calls, configure the JVM by setting the com.ibm.websphere.sendredirect.compatibility property to a true or false value.

#### Steps for this task

1. Access the settings page for a property of the JVM.
  - a. Click **Servers > Application Servers** in the console navigation tree.
  - b. On the Application Server page, click on the name of the server whose JVM settings you want to configure.
  - c. On the settings page for the selected application server, click **Process Definition**.
  - d. On the Process Definition page, click **Java Virtual Machine**.
  - e. On the Java Virtual Machine page, click **Custom Properties**.
  - f. On the Properties page, click **New**.
2. On the settings page for a property, specify a name of com.ibm.websphere.sendredirect.compatibility and either true or false for the value, then click **OK**.
3. Click **Save** on the console taskbar.
4. Stop the application server and then restart the application server.

---

## Preparing to host applications

The default application server and a set of default resources are available to help you begin quickly. Suppose you choose instead to configure a new server and set of resources. Here is what you need to do in order to set up a run-time environment to support applications.

#### Steps for this task

1. Create an application server.
2. Create a virtual host.
3. Configure a Web container.
4. Configure an EJB container.
5. Create resources for data access.
6. Create a JDBC provider and data source.
7. Create a URL and URL provider.
8. Create a JMS destination, connection, and provider.
9. Create a JavaMail session.
10. Create resources for session support.
11. Configure a Session Manager.

---

## Application servers: Resources for learning

Use the following links to find relevant supplemental information about configuring application servers. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.


View links to additional information about:

- Programming model and decisions
- Programming instructions and examples
- Programming specifications
- Administration



#### **Programming model and decisions**

-  **Exploiting the Java Virtual Machine**  
(<http://www.develop.com/downloads/DevWPJav.pdf>)





#### **Programming instructions and examples**

-  **WebSphere Application Server education**  
(<http://www.ibm.com/software/webservers/learn/>)

#### **Programming specifications**

-  **The Java™ Virtual Machine Specification, Second Edition**  
(<http://java.sun.com/docs/books/vmspec/>)
-  **Sun's technology forum for the Java™ Virtual Machine Specification**  
(<http://forum.java.sun.com/forum.jsp?forum=37>)

#### **Administration**

-  **IBM WebSphere Administration** (<http://www.mcgraw-hill.co.uk/html/0072223154.html>)
-  **Listing of all IBM WebSphere Application Server Redbooks**  
(<http://publib-b.boulder.ibm.com/Redbooks.nsf/Portals/WebSphere>)
-  **IBM WebSphere V4.0 Advanced Edition Handbook**  
(<http://www.redbooks.ibm.com/abstracts/sg246176.html>)
-  **WebSphere 4.0 Installation and Configuration on the IBM iSeries Server**  
(<http://publib-b.boulder.ibm.com/Redbooks.nsf/9445fa5b416f6e32852569ae006bb65f/7b1a07251256f08b85256b750067aee1?OpenDocument>)
-  **Redbook on Backing up WebSphere Application Server with Tivoli Storage Management** (<http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/redp0149.html?Open>)



---

## Chapter 3. Managing Object Request Brokers

Default property values are set when the product is started and the Java Object Request Broker (ORB) service is initialized. These properties control the run-time behavior of the ORB and can also affect the behavior of product components that are tightly integrated with the ORB, such as security. It might be necessary to modify some ORB settings under certain conditions.

In every request/response exchange, there is a client-side ORB and a server-side ORB. It is important that the ORB properties be set for both sides as necessary.

After an ORB instance has been established in a process, changes to ORB properties do not affect the behavior of the running ORB instance. The process must be stopped and restarted in order for the modified properties to take effect.

The following steps are to be performed only as needed.

### Steps for this task

1. **(Optional)** Adjust timeout settings to improve handling of network failures. Before making these adjustments, be sure to read "ORB tuning guidelines."
2. **(Optional)** Adjust (thread-pool settings) used by the ORB for handling IIOP connections.
3. If problems with the ORB arise, determine the problem. For help in troubleshooting, look at the ORB communications trace.

---

## Object Request Brokers

An Object Request Broker (ORB) manages the interaction between clients and servers, using the Internet InterORB Protocol (IIOP). It enables clients to make requests and receive responses from servers in a network-distributed environment.

The ORB provides a framework for clients to locate objects in the network and call operations on those objects as if the remote objects were located in the same running process as the client, providing location transparency. The client calls an operation on a local object, known as a stub. Then the stub forwards the request to the desired remote object, where the operation is run and the results are returned to the client.

The client-side ORB is responsible for creating an IIOP request that contains the operation and any required parameters, and for sending the request on the network. The server-side ORB receives the IIOP request, locates the target object, invokes the requested operation, and returns the results to the client. The client-side ORB demarshals the returned results and passes the result to the stub, which, in turn, returns to the client application, as if the operation had been run locally.

This product uses an ORB to manage communication between client applications and server applications as well as communication among product components. During product installation, default property values are set when the ORB is initialized. These properties control the run-time behavior of the ORB and can also

affect the behavior of product components that are tightly integrated with the ORB, such as security. This product does not support the use of multiple ORB instances.

---

## Object Request Broker tuning guidelines

If Web clients that access Java applications running in the product environment are consistently experiencing problems with their requests, and the problem cannot be traced to other sources and addressed through other solutions, consider setting an Object Request Broker (ORB) time-out value and adjusting it for your environment.

- Web browsers vary in their language for indicating that they have timed out. Usually, the problem is announced as a connection failure or no-path-to-server message.
- Aim to set an ORB time-out value to less than the time after which a Web client eventually times out. Because it can be difficult to tell how long Web clients will wait before timing out, setting an ORB time-out requires some experimentation. Another difficulty is that the ideal testing environment features some simulated network failures for testing the proposed setting value.
- Empirical results from limited testing indicate that 30 seconds is a reasonable starting value. Mainly, you need to ensure that the setting is not too low. To fine-tune the setting, find a value that is not too low. Then gradually decrease the setting until reaching the threshold at which the value becomes too low. Set the value a little above the threshold.
- When an ORB time-out value is set too low, the symptom is numerous CORBA 'NO\_RESPONSE' exceptions, which occur even for some requests that should have been valid. If requests that should have been successful (for example, the server is not down) are being lost or refused, the value is likely to be too low.

**Note:** Do not adjust an ORB time-out value unless experiencing a problem, because configuring a value that is inappropriate for the environment can itself create a problem. If you set the value, experimentation might be needed to find the correct value for the particular environment. Configuring an incorrect value can produce results worse than the original problem.

You can adjust time-out intervals for the product's Java ORB through the following administrative settings:

- **Request timeout**, the number of seconds to wait before timing out on most pending ORB requests if the network fails
- **Locate request timeout**, the number of seconds to wait before timing out on a locate-request message

You can also improve performance by setting the `com.ibm.CORBA.numJNIReaders` system property through a command-line script. This property specifies the number of threads to be shared for request handling when the native selector mechanism is enabled. The default value of this property is 2. Valid settings for this property range from 0 to 2147483647.

---

## Object Request Broker service settings in administrative console

Use this page to configure the Java Object Request Broker (ORB) service.

To view this administrative console page, click **Servers > Application Servers > *serverName* > ORB Service**.

## Request timeout

Specifies the number of seconds to wait before timing out on a request message.

For use in command-line scripting, the full name of this system property is `com.ibm.CORBA.RequestTimeout`.

Data type	int
Units	Seconds
Default	180
Range	0 to 300

## Request retries count

Specifies the number of times that the ORB attempts to send a request if a server fails. Retrying sometimes enables recovery from transient network failures.

For use in command-line scripting, the full name of this system property is `com.ibm.CORBA.requestRetriesCount`.

Data type	int
Default	1
Range	1 to 10

## Request retries delay

Specifies the number of milliseconds between request retries.

For use in command-line scripting, the full name of this system property is `com.ibm.CORBA.requestRetriesDelay`.

Data type	int
Units	Milliseconds
Default	0
Range	0 to 60

## Connection cache maximum

Specifies the largest number of connections allowed to occupy the connection cache for the service.

Data type	Integer
Units	Connections
Default	240

## Connection cache minimum

Specifies the smallest number of connections allowed to occupy the connection cache for the service.

Data type	Integer
Units	Connections
Default	100

## ORB tracing

Enables the tracing of ORB GIOP messages.

This setting affects two system properties: `com.ibm.CORBA.Debug` and `com.ibm.CORBA.CommTrace`. If you set these properties through command-line scripting, you must set both to `true` in order to enable the tracing of GIOP messages.

Data type	Boolean
Default	Not enabled ( <code>false</code> )

## Locate request timeout

Specifies the number of seconds to wait before timing out on a `LocateRequest` message.

For use in command-line scripting, the full name of this system property is `com.ibm.CORBA.LocateRequestTimeout`.

Data type	int
Units	Seconds
Default	180
Range	0 to 300

## Force tunneling

Controls how the client ORB attempts to use HTTP tunneling.

For direct access, the full name of this property is `com.ibm.CORBA.ForceTunnel`.

Data type	String
Default	NEVER
Range	Valid values are ALWAYS, NEVER, or WHENREQUIRED.

Additional information about valid values follows:

### ALWAYS

Use HTTP tunneling immediately, without trying TCP connections first.

### NEVER

Disable HTTP tunneling. If a TCP connection fails, a CORBA system exception (`COMM_FAILURE`) is thrown.

### WHENREQUIRED

Use HTTP tunneling if TCP connections fail.

## Tunnel agent URL

Specifies the URL of the servlet used to support HTTP tunneling.

This must be a properly formed URL, such as `http://w3.mycorp.com:81/servlet/com.ibm.CORBA.services.IIOPTunnelServlet` or, for applets, `http://applethost:port/servlet/com.ibm.CORBA.services.IIOPTunnelServlet`. This field is required if **HTTP tunneling** is set.

For use in command-line scripting, the full name of this system property is `com.ibm.CORBA.TunnelAgentURL`.

## Pass by reference

When enabled, this specifies that the ORB is to pass parameters by reference instead of by value, which bypasses a copy operation. Enable this property with caution, because unexpected behavior might occur.

For use in command-line scripting, the full name of this system property is `com.ibm.CORBA.iiop.noLocalCopies`.

Data type	Boolean
Default	Not enabled (false)

---

## Object Request Broker service settings that can be added to the administrative console

Use the Properties page to set and monitor settings associated with the Java Object Request Broker (ORB) service that do not appear on the main settings page by default.

To view that administrative console page, click **Servers > Application Servers > *serverName* > ORB Service > Custom Properties**.

To add properties to this page, click **New** and enter at least a name and value for the property. Then click **Apply**. When you are finished entering properties, click **OK**.

The page might already include Secure Socket Layer (SSL) properties that were added during product setup. A list of additional properties associated with the Java ORB service follows:

### **com.ibm.CORBA.BootstrapHost**

Specifies the DNS host name or IP address of the machine on which initial server contact for this client resides. This setting is deprecated and will be removed in a future release. For a command-line or programmatic alternative, see "Programming tips for the Java Object Request Broker service."

### **com.ibm.CORBA.BootstrapPort**

Specifies the port to which the ORB connects for bootstrapping. In other words, the port of the machine on which the initial server contact for this client is listening. The default value is 2809. This setting is deprecated and will be removed in a future release. For a command-line or programmatic alternative, see "Programming tips for the Java Object Request Broker service."

### **com.ibm.CORBA.FragmentSize**

Specifies the size of GIOP fragments used by the ORB. If the total size of a request exceeds the set value, the ORB breaks up and sends multiple fragments until the entire request is sent. The default value is 1024 bytes. The valid range is from 64 to the largest value of the Java int type that is divisible by 8.

### **com.ibm.CORBA.ListenerPort**

Specifies the port on which this server listens for incoming requests. The

setting of this property is valid only for client-side ORBs. The default value is the next available system-assigned port number. The valid range is 0 to 2147483647.

**com.ibm.CORBA.LocalHost**

Specifies the host name or IP address of the system on which the server ORB is running. The setting of this property is valid only for client-side ORBs. Otherwise, the ORB obtains a value at run time by calling `InetAddress.getLocalHost().getHostAddress()`.

**com.ibm.CORBA.ServerSocketQueueDepth**

The property changes the maximum queue length for server incoming TCP/IP connection requests. If a connection requests arrives when the queue is full, the connection is refused. The valid range is between 50 and the maximum Java int value. The default value is 50.

**com.ibm.CORBA.ShortExceptionDetails**

If set to any value, this specifies that the exception detail message that is returned whenever the server ORB encounters a CORBA system exception is to contain a short description of the exception as returned by the `toString()` method of `java.lang.Throwable`. Otherwise, the message contains the complete stack trace as returned by the `printStackTrace()` method of `java.lang.Throwable`.

If needed for locale support, you can also set and monitor properties for codeset conversion. For details, see "Character codeset conversion support for the Java Object Request Broker service."

---

## Object Request Broker communications trace

The Object Request Broker (ORB) communications trace, typically referred to as *CommTrace*, contains the sequence of General InterORB Protocol (GIOP) messages sent and received by the ORB during application execution. It might be necessary to understand the low-level sequence of client-to-server or server-to-server interactions during problem determination. This article uses trace entries from log examples to explain the contents of the log and help you understand the interaction sequence. It focuses only in the GIOP messages and does not discuss in detail additional trace information that appears when intervening with the GIOP-message boundaries.

### Location

When ORB tracing is enabled, this information is placed in `install_root/logs/trace`.

### Usage notes

- Is this file read-only?  
Yes
- Is this file updated by a product component?  
This file is updated by the administrative function.
- How and when are the contents of this file used?  
You use this file to localize and resolve ORB-related problems.

### How to interpret the output

The following sections refer to sample log output found later in this topic.

### Identifying information

The start of a GIOP message is identified by a line which contains either "OUT GOING:" or "IN COMING:" depending on whether the message is sent or received by the process that is being traced.

Following the identifying line entry is a series of items, formatted for convenience, with information extracted from the raw message that identify the endpoints in this particular message interaction. See lines 3-13 in both examples. The formatted items include the following:

- GIOP message type (line 3)
- Date and time that message was recorded (line 4)
- Information useful in uniquely identifying the thread in execution when the message was recorded, with other thread-specific information (line 5, broken for publication in the reply example)
- Local and remote TCP/IP ports used for the interaction (lines 6 through 9)
- GIOP version, byte order, whether the message is a fragment, and message size (lines 10 through 13)

### Request ID, response expected and reply status

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

- Lines 15-17 in the request example show the request ID, followed by an indication to the receiving endpoint that a response is expected (CORBA allows sending of one-way requests for which a response is not expected.)
- Line 15 in Sample Log Entry - GIOP Reply shows the request ID; line 33 shows the reply status received after completing the previously sent request.

### Object Key

Lines 18-20 in the request example show the object key, the internal representation used by the ORB during execution to identify and locate the target object intended to receive the request message. Object keys are not standardized.

### Operation

Line 21 in the request example shows the name of the operation to be executed by the target object in the receiving endpoint. In this example, the specific operation requested is named `_get_value`.

### Service context information

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

Lines 22-41 in the request example illustrate typical service context entries. There are three service contexts in the request message, as shown in line

22. The ID, length of data, and raw data for each service context is printed next. Lines 23-25 show an IBM-proprietary context, as indicated by the ID 0x49424D12. Lines 26-41 show two standard service contexts, identified by ID 0x6 (line 26) and 0x1 (line 39).

Lines 16-32 in the Sample Log Entry - GIOP Reply illustrate two service contexts, one IBM-proprietary (line 17) and one standardized (line 20).

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

#### Data offset

Line 42 in the request example shows the offset, relative to the beginning of the GIOP message, where the remainder body of the request or reply message is located. This portion of the message is specific to each operation and varies from operation to operation. Therefore, it is not formatted, as the specific contents are not known by the ORB. The offset is printed as an aid to quickly locating the operation-specific data in the raw GIOP message dump, which follows the data offset.

#### Raw GIOP message dump

Starting at line 45 in the request example and line 36 in Sample Log Entry - GIOP Reply, a raw dump of the entire GIOP message is printed in hexadecimal format. Request messages contain the parameters required by the given operation and reply messages contain the return values and content of output parameters as required by the given operation. For brevity, not all of the raw data has been included in the figures.

#### Sample Log Entry - GIOP Request

```
1. OUT GOING:

3. Request Message
4. Date:          April 17, 2002 10:00:43 PM CDT
5. Thread Info:  P=842115:0=1:CT
6. Local Port:   1243 (0x4DB)
7. Local IP:    jdoe.austin.ibm.com/192.168.1.101
8. Remote Port: 1242 (0x4DA)
9. Remote IP:   jdoe.austin.ibm.com/192.168.1.101
10. GIOP Version: 1.2
11. Byte order:  big endian
12. Fragment to follow: No
13. Message size: 268 (0x10C)
--
15. Request ID:      5
16. Response Flag:  WITH_TARGET
17. Target Address:  0
18. Object Key:     length = 24 (0x18)
                   4B4D4249 00000010 BA4D6D34 000E0008
                   00000000 00000000
21. Operation:      _get_value
22. Service Context: length = 3 (0x3)
23. Context ID:    1229081874 (0x49424D12)
24. Context data:  length = 8 (0x8)
                   00000000 13100003
26. Context ID:    6 (0x6)
```



```

27. Context data: length = 164 (0xA4)
    00000000 00000028 49444C3A 6F6D672E
    6F72672F 53656E64 696E6743 6F6E7465
    78742F43 6F646542 6173653A 312E3000
    00000001 00000000 00000068 00010200
    0000000E 3139322E 3136382E 312E3130
    310004DC 00000018 4B4D4249 00000010
    BA4D6D69 000E0008 00000000 00000000
    00000002 00000001 00000018 00000000
    00010001 00000001 00010020 00010100
    00000000 49424D0A 00000008 00000000
    13100003
39. Context ID: 1 (0x1)
40. Context data: length = 12 (0xC)
    00000000 00010001 00010100
42. Data Offset: 118

45. 0000: 47494F50 01020000 0000010C 00000005  GIOP.....
46. 0010: 03000000 00000000 00000018 4B4D4249  ....KMBI
47. 0020: [remainder of message body deleted for brevity]

```

### Sample Log Entry - GIOP Reply

```

1. IN COMING:

3. Reply Message
4. Date: April 17, 2002 10:00:47 PM CDT
5. Thread Info: RT=0:P=842115:O=1:com.ibm.rmi.transport.TCPTransportConnection
5a. remoteHost=192.168.1.101 remotePort=1242 localPort=1243
6. Local Port: 1243 (0x4DB)
7. Local IP: jdoe.austin.ibm.com/192.168.1.101
8. Remote Port: 1242 (0x4DA)
9. Remote IP: jdoe.austin.ibm.com/192.168.1.101
10. GIOP Version: 1.2
11. Byte order: big endian
12. Fragment to follow: No
13. Message size: 208 (0xD0)
--
15. Request ID: 5
16. Service Context: length = 2 (0x2)
17. Context ID: 1229081874 (0x49424D12)
18. Context data: length = 8 (0x8)
    00000000 13100003
20. Context ID: 6 (0x6)
21. Context data: length = 164 (0xA4)
    00000000 00000028 49444C3A 6F6D672E
    6F72672F 53656E64 696E6743 6F6E7465
    78742F43 6F646542 6173653A 312E3000
    00000001 00000000 00000068 00010200
    0000000E 3139322E 3136382E 312E3130
    310004DA 00000018 4B4D4249 00000010
    BA4D6D34 000E0008 00000001 00000000
    00000002 00000001 00000018 00000000
    00010001 00000001 00010020 00010100
    00000000 49424D0A 00000008 00000000
    13100003
33. Reply Status: NO_EXCEPTION

36. 0000: 47494F50 01020001 000000D0 00000005  GIOP.....
37. 0010: 00000000 00000002 49424D12 00000008  ....IBM.....
38. 0020: [remainder of message body deleted for brevity]

```

---

## Client-side programming tips for the Java Object Request Broker service

This article includes programming tips for applications that communicate with the client-side Object Request Broker (ORB) that is part of the Java ORB service.

### Resolution of initial references to services

Client applications can use the properties *ORBInitRef* and *ORBDefaultInitRef* to configure the network location that the Java ORB service uses to find a service such as naming. Once set, these properties are included in the parameters used to initialize the ORB, as follows:

```
org.omg.CORBA.ORB.init(java.lang.String[] args,  
                      java.util.Properties props)
```

You can set these properties in client code or by command-line argument. It is possible to specify more than one service location by using multiple *ORBInitRef* property settings (one for each service), but only a single value for *ORBDefaultInitRef* may be specified. For more information about the two properties and the order of precedence that the ORB uses to locate services, read the CORBA/IIOP specification, cited in "Resources for learning."

For setting in client code, these properties are *com.ibm.CORBA.ORBInitRef.service\_name* and *com.ibm.CORBA.ORBDefaultInitRef*, respectively. For example, to specify that the naming service (NameService) is located in *sample.server.com* at port 2809, set the *com.ibm.CORBA.ORBInitRef.NameService* property to *corbaloc::sample.server.com:2809/NameService*.

For setting by command-line argument, these properties are *-ORBInitRef* and *-ORBDefaultInitRef*, respectively. To locate the same naming service specified previously, use the following Java command (split here for publication only):

```
java program -ORBInitRef  
             NameService=corbaloc::sample.server.com:2809/NameService
```

After these properties have been set for services supported by the ORB, J2EE applications obtain the initial reference to a given service by calling the *resolve\_initial\_references* function on the ORB as defined in the CORBA/IIOP specification.

### Preferred API for obtaining an ORB instance

For J2EE applications, you can use either of the following approaches. However, it is strongly recommended that you use the JNDI approach to ensure that the same ORB instance is used throughout the client application; you will avoid the unintended inconsistencies that might occur when different ORB instances are used.

**JNDI approach:** For J2EE applications (including enterprise beans, J2EE clients and servlets), you can obtain an ORB instance by creating a JNDI InitialContext object and looking up the ORB under the name *java:comp/ORB*, as follows:

```
javax.naming.Context ctx = new javax.naming.InitialContext();  
org.omg.CORBA.ORB orb =  
    (org.omg.CORBA.ORB) javax.rmi.PortableRemoteObject.narrow(ctx.lookup(  
    "java:comp/ORB"), org.omg.CORBA.ORB.class);
```

The ORB instance obtained using JNDI is a singleton object, shared by all J2EE components running in the same Java virtual machine process.

**CORBA approach:** Because thin-client applications do not run in a J2EE container, they cannot use JNDI interfaces to look up the ORB. In this case, you can obtain an ORB instance by using CORBA programming interfaces, as follows:

```
java.util.Properties props = new java.util.Properties();
java.lang.String[] args = new java.lang.String[0];
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, props);
```

In contrast to the JNDI approach, the CORBA specification requires that a new ORB instance be created each time the ORB.init method is called. If necessary to change the ORB's default settings, you can add ORB property settings to the Properties object that is passed in the ORB.init() call.

The use of com.ibm.ejs.oa.EJSORB.getORBInstance(), supported in previous releases of this product, has been deprecated.

### API restrictions with sharing an ORB instance among J2EE application components

For performance reasons, it often makes sense to share a single ORB instance among components in a J2EE application. As required by the J2EE Specification, Version 1.3, all web and EJB containers provide an ORB instance in the JNDI namespace as java:comp/ORB. Each container can share this instance among application components but is not required to. For proper isolation between application components, application code must comply with the following restrictions:

- Do not call the ORB shutdown method
- Do not call org.omg.CORBA\_2\_3.ORB methods register\_value\_factory or unregister\_value\_factory

In addition, an ORB instance should not be shared among application components in different J2EE applications.

### Required use of rmic and idlj shipped with the IBM Developer Kit

The Java Runtime Environment (JRE) used by this product includes the tools **rmic** and **idlj**. You use the tools to generate Java language bindings for the CORBA/IIOP protocol.

During product installation, the tools are installed in the directory *installation\_root*/java/ibm\_bin, where *installation\_root* is the installation directory for the product. Versions of these tools included with Java development kits in \$JAVA\_HOME/bin other than the IBM Developer Kit installed with this product are incompatible with this product.

When you install this product, the directory *installation\_root*/java/ibm\_bin is included in the \$PATH search order to enable use of the rmic and idlj scripts provided by IBM. Because the scripts are in *installation\_root*/java/ibm\_bin instead of the JRE standard location *installation\_root*/java/bin, it is unlikely that you will overwrite them when applying maintenance to a JRE not provided by IBM.

In addition to the rmic and idlj tools, the JRE also includes Interface Definition Language (IDL) files. The files are based on those defined by the Object

Management Group (OMG) and can be used by applications that need an IDL definition of selected ORB interfaces. The files are placed in the *installation\_root/java/ibm\_lib* directory.

Before using either the *rmic* or *idlj* tool, ensure that the *installation\_root/java/ibm\_bin* directory is included in the proper PATH variable search order in the environment. If your application will use IDL files in the *installation\_root/java/ibm\_lib* directory, also ensure that the directory is included in the PATH variable.

---

## Character codeset conversion support for the Java Object Request Broker service

The CORBA/IIOP specification defines a framework for negotiation and conversion of character codesets used by the Java Object Request Broker (ORB) service. This product supports the framework and provides the following system properties for modifying the default settings:

### **com.ibm.CORBA.ORBCharEncoding**

Specifies the name of the native codeset that the ORB is to use for character data (referred to as NCS-C in the CORBA/IIOP specification). By default, the ORB uses UTF8. (In contrast, the default value for versions 3.5.x and 4.0.x of this product was IS08859\_1, also known as Latin-1.) Valid codeset values for this property are shown in the table that follows this list; values that are valid only for ORBWCharDefault are indicated.

### **com.ibm.CORBA.ORBWCharDefault**

Specifies the default codeset that the ORB is to use for transmission of wide character data when no codeset for wide character data is found in the tagged component in the Interoperable Object Reference (IOR) or in the GIOP service context. By default, the ORB uses UCS2. The only valid codeset values for this property are UCS2 or UTF16.

The CORBA codeset negotiation/conversion framework specifies the use of codeset registry IDs as defined in the Open Software Foundation (OSF) codeset registry. The ORB translates the Java *file.encoding* names shown in the following table to the corresponding OSF registry IDs. These IDs are then used by the ORB in the IOR Codeset tagged component and GIOP Codeset service context as specified in the CORBA/IIOP specification.

Java name	OSF registry ID	Comments
ASCII	0x00010020	
ISO8859_1	0x00010001	
ISO8859_2	0x00010002	
ISO8859_3	0x00010003	
ISO8859_4	0x00010004	
ISO8859_5	0x00010005	
ISO8859_6	0x00010006	
ISO8859_7	0x00010007	
ISO8859_8	0x00010008	
ISO8859_9	0x00010009	
ISO8859_15_FDIS	0x0001000F	

Java name	OSF registry ID	Comments
Cp1250	0x100204E2	
Cp1251	0x100204E3	
Cp1252	0x100204E4	
Cp1253	0x100204E5	
Cp1254	0x100204E6	
Cp1255	0x100204E7	
Cp1256	0x100204E8	
Cp1257	0x100204E9	
Cp943C	0x100203AF	
Cp943	0x100203AF	
Cp949C	0x100203B5	
Cp949	0x100203B5	
Cp1363C	0x10020553	
Cp1363	0x10020553	
Cp950	0x100203B6	
Cp1381	0x10020565	
Cp1386	0x1002056A	
EUC_JP	0x00030010	
EUC_KR	0x0004000A	
EUC_TW	0x00050010	
Cp964	0x100203C4	
Cp970	0x100203CA	
Cp1383	0x10020567	
Cp33722C	0x100283BA	
Cp33722	0x100283BA	
Cp930	0x100203A2	
Cp1047	0x10020417	
UCS2	0x00010100	Valid only for ORBWCharDefault
UTF8	0x05010001	
UTF16	0x00010109	Valid only for ORBWCharDefault

For more information, read the CORBA/IIOP specification, cited in "Resources for learning."

---

## Object Request Brokers: Resources for learning

Use the following links to find relevant supplemental information about Object Request Brokers (ORBs). The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.


These links are provided for convenience. Often, the information is not specific to this product but is useful all or in part for understanding the product. When

possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.



View links to additional information about:

- Planning, business scenarios, and IT architecture
- Administration
- Programming specifications

#### **Planning, business scenarios, and IT architecture**

-  **CORBA FAQ** (<http://www.omg.org/gettingstarted/corbafaq.htm>)  
Getting started with object request brokers and CORBA.

#### **Administration**

-  **IANA Character Set Registry** (<http://www.iana.org/assignments/character-sets>)  
This contains a list of all valid character encoding schemes.
-  **WebSphere Interoperability between Versions 3.5.x and 4.0.x**  
([http://www7b.boulder.ibm.com/wsdd/library/techarticles/0202\\_sundman/sundman.html](http://www7b.boulder.ibm.com/wsdd/library/techarticles/0202_sundman/sundman.html))  
This WebSphere Developer Domain article by Joel Sundman and Matt Kelm (February 2002, updated May 2002) is not directly related to the Java ORB service, but it touches upon ORB-related issues.

#### **Programming specifications**

-  **Catalog Of OMG CORBA/IIOP Specifications**  
([http://www.omg.org/technology/documents/corba\\_spec\\_catalog.htm](http://www.omg.org/technology/documents/corba_spec_catalog.htm))