IBM WebSphere Application Server Enterprise
Version 5.0.2

Servers

Note

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

Compilation date: November 18, 2003

Contents

Trademarks and service marks

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 as follows.
- "Configuring transports" on page 18
- "Developing custom services" on page 24
- "Defining application server processes" on page 27 from the command line
- Settings that improve the use of "Java virtual machines (JVMs)" on page 32.

See Chapter 2, "Configuring application servers," on page 3.

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

See (backup clusters) for information on backup cluster support. This feature allows the client to continue functioning when all cluster members of the primary cluster are not available.

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.

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. On the Network Deployment product, you can also create a new application server when you (add a cluster member to a server cluster). 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 it 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 base WebSphere Application Server 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 (dmgr) process. From the base product or the Network Deployment product, use the wasadmin tool or command line tools such as startServer and stopServer to manage the Application Server.

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. The Network Deployment product also shows the status of the Application Servers. The status indicates whether a server is started, stopped, or unavailable.

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

To view product information for an Application Server:

a. Verify that the Application Server is running.
b. Display 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 Application Server, the version and build levels for the products, the build dates, and any interim fixes applied to the Application Server.

3. Create an Application Server. Click **New** and follow the instructions on the Create New Application Server page.
4. Start your Application Server.
5. Monitor the running of Application Servers.
6. Stop your Application Server.
7. **(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 an Application Server to delete it.
   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

Application Class loader Policy

Specifies whether to use a single class loader to load all applications or to use a different class loader for each application.

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

Data type	String
Default	MULTIPLE

Application Classloading Mode

Specifies whether the class loader should search in the parent class loader or in the application class loader first to load a class. The standard for JDK class loaders and WebSphere class loaders is PARENT_FIRST. By specifying PARENT_LAST, your application can override classes contained in the parent class loader, but this action can potentially result in ClassCastException or LinkageErrors if you have mixed use of overriden classes and non-overriden classes.

The options are PARENT_FIRST and PARENT_LAST. The default is to search in the parent class loader before searching in the application class loader to load a class.

Data type	String
Default	PARENT_FIRST

Short name

Specifies the short name of the server.

The name is 1-8 characters, alpha-numeric or national language. It cannot start with a numeric.

The system assigns a cell-unique, default short name.

Unique Id

Specifies the unique ID of this server.

The unique ID property is read only. The system automatically generates the value.

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	*host_name*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 one of the following paths:

- **Servers > Application Servers >** *server_name* **> End Points >** *end_point_name*
- **Servers > JMS Servers >** *server_name* **> Security Port Endpoint**
- **Servers > JMS 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

Custom 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 one of the following paths:

- **Servers > Application Servers >** *server_name* **> Custom Properties**
- **Servers** > **JMS 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.

Required

Specifies whether the value field requires a value. If this box is checked, you must provide a value.

Valid Expression

Custom 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 one of the following paths:

- **Servers > Application Servers >** *server_name* **> Custom Properties >** *property_name*
- **Servers** > **JMS 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. Requests are sent to the server through any of the HTTP transports. 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.

If your Tivoli Performance Viewer shows the Percent Maxed metric to remain consistently in the double digits, consider increasing the Maximum size. The Percent Maxed metric indicates the amount of time that the configured threads are used. If there are several simultaneous clients connecting to the server-side ORB, increase the size to support up to 1000 clients.

Data type	Integer
Default	50
Recommended	50 (25 on Linux systems)

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 Allow thread allocation beyond maximum thread size or Not enabled.

Starting servers

Starting a server starts a new server process based on the process definition settings of the current server configuration. The node agent for the node on which the Application Server resides must be running before you can start the Application Server.

Note: After you start an Application Server, other processes might not discover the running Application Server immediately. Application Servers are discovered by the node agent. Node agents are discovered by the deployment manager. Node agents usually can discover local Application Servers quickly but it can take a deployment manager from 2 to 60 seconds to discover a node agent.

There are several options for starting an Application Server:

Steps for this task

1. **(Optional)** Use the startServer command to start an Application Server from the command line.

   If the node agent for the node on which the Application Server resides is not running, run the startNode command and then run the startServer command command.
2. **(Optional)** Start an Application Server using the administrative console.
   a. Click **Servers > Application Servers** from the administrative console navigation tree to go to the Application Server page.
   b. If the node agent for the node on which the Application Server resides is not running, click **Restart** or **Restart all Servers on Node** on the Node Agents page to start the node agent.

      If the node agent does not start, run the startNode command and then run the startServer command. Once a node agent completely stops running and remains stopped, you cannot remotely start the node agent from the Node Agents page. You must run the startNode command to start the node agent on the node where it runs.
   c. Place a checkmark in the check box beside the Application Server to start and click **Start**.
   d. View the **Status** value and any messages or logs to see whether the Application Server starts.
3. **(Optional)** Start an Application Server for tracing and debugging.

   To start the Application Server with standard Java debugging enabled:
   a. 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**.
   b. 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**.
   c. Save the changes to a configuration file.
   d. Stop the Application Server.
   e. Start the Application Server again as described previously.

Running an Application Server with a non-root user ID and the nodeagent as root

Use this task to configure an Application Server to run as non-root.

By default, WebSphere Application Server on UNIX platforms uses the root user ID to run Application Servers. You can use a non-root user ID to run Application Servers.

If global security is enabled, it is not recommended that the Local OS be used for user registry. In general, using the Local OS user registry requires that all processes run as root. Refer to ″Local operating system user registries″ (not in this document) for details.

Using a non-root user ID to run Application Servers can be done by setting all the Application Servers to run under the same operating system group. If running the WebSphere JMS provider, add the `jmsserver` server to the `mqm` group to allow `jmsserver` to start the message queue. If not running `jmsserver`, you can use a group other than `mqm` in the following steps:

Steps for this task

1. Log on as root.
2. Create the `was1` user ID to be used to run the Application Server.
3. Add users `root` and `was1` to the `mqm` group.
4. Reboot the machine.
5. Configure Application Server properties for the `root` and `was1` users.

   Use the administrative console to complete the following steps:

   a. Define the nodeagent to run as a root process.

      Click **System Management > Node Agents > nodeagent** *(for the node)* **> Process Definition > Process Execution** and change these values:

Property	Value
Run As User	root
Run As Group	mqm
UMASK	002

   b. Define each Application Server to run as a `was1` process. Substitute the name of each server for `server1`.

      Click **Servers > Application Servers > server1 > Process Definition > Process Execution** and change these values:

Property	Value
Run As User	was1
Run As Group	mqm
UMASK	002

   c. If running the WebSphere JMS provider, define the `jmsserver` process to run as a `root` process.

      Click **JMS Servers > jmsserver** *(for the node)* **> Process Definition > Process Execution** and change these values:

Property	Value
Run As User	root
Run As Group	mqm
UMASK	002

6. Save and synchronize.
7. Stop all servers, including the server1 and jmsserver servers.

   Use the **stopserver** command:

   ```
   stopserver server1
   stopserver jmsserver
   ```
8. Stop the node.

   Use the **stopnode** command:

   ```
   stopnode
   ```
9. As root, use operating system tools to change file permissions.

   The following examples assume that the WebSphere Application Server installation root directory is `/opt/WebSphere/AppServer`:

   ```
       chgrp mqm /opt/WebSphere
       chgrp mqm /opt/WebSphere/AppServer
       chgrp -R mqm /opt/WebSphere/AppServer/config
       chgrp -R mqm /opt/WebSphere/AppServer/logs
       chgrp -R mqm /opt/WebSphere/AppServer/wstemp
       chgrp -R mqm /opt/WebSphere/AppServer/installedApps
       chgrp -R mqm /opt/WebSphere/AppServer/temp
       chgrp -R mqm /opt/WebSphere/AppServer/tranlog
       chgrp -R mqm /opt/WebSphere/AppServer/cloudscape50
       chgrp -R mqm /opt/WebSphere/AppServer/cloudscape51
       chgrp -R mqm /opt/WebSphere/AppServer/bin/DefaultDB
       chmod g+w /opt/WebSphere
       chmod g+w /opt/WebSphere/AppServer
       chmod -R g+w  /opt/WebSphere/AppServer/config
       chmod -R g+w /opt/WebSphere/AppServer/logs
       chmod -R g+w /opt/WebSphere/AppServer/wstemp
       chmod -R g+w /opt/WebSphere/AppServer/installedApps
       chmod -R g+w /opt/WebSphere/AppServer/temp
       chmod -R g+w /opt/WebSphere/AppServer/tranlog
       chmod -R g+w /opt/WebSphere/AppServer/cloudscape50
       chmod -R g+w /opt/WebSphere/AppServer/cloudscape51
       chmod -R g+w /opt/WebSphere/AppServer/bin/DefaultDB
   ```
10. Start the node, the jmsserver, and all Application Servers.

    Start the `nodeagent` and the `jmsserver` from root. Start each Application Server from the `was1` user.
11. If running the WebSphere JMS provider, verify that the MQ queue is running.

    Run the **dspmq** command:

    ```
    dspmq
    ```

    The name of the queue is `WAS_wasnode_jmsserver`.

Results

You can start an Application Server from a non-root user.

Running an Application Server and nodeagent with a non-root user ID

By default, each base Application Server node on Linux and UNIX platforms uses the root user ID to run the nodeagent process, the jmsserver process, and all Application Server processes. You can run the nodeagent, the jmsserver, and all Application Server processes under the same non-root user and user group.

If global security is enabled, the user registry must not be Local OS. Using the Local OS user registry requires the nodeagent to run as root.

Using the same non-root user and user group gives the nodeagent process the operating system permissions to start all other server processes. If using the WebSphere JMS provider, the user group must be `mqm` for the jmsserver to start the message queue. If you are not using the WebSphere JMS provider, you can specify a user group other than `mqm`.

For the steps that follow, assume that:
- `wasadmin` is the user to run all servers
- `wasnode` is the node name
- `wascell` is the cell name
- `mqm` and `mqbrkrs` are user groups associated with the WebSphere JMS provider
- `server1` is the Application Server
- `/opt/WebSphere/Appserver` is the installation root
- `jmsserver` exists because you are using the WebSphere JMS provider

To configure a user ID to run the nodeagent and all server processes, complete the following steps:

Steps for this task

1. Log on as root.
2. Create user `wasadmin` with primary group `mqm`.

   Also add user `wasadmin` to group `mqbrkrs` if you are running the WebSphere JMS provider.
3. Reboot the machine.
4. Define the nodeagent to run as a `wasadmin` process.

   Click **System Management > Node Agents > nodeagent** *(for the node)* **> Process Definition > Process Execution** and change these values:

Property	Value
Run As User	wasadmin
Run As Group	mqm
UMASK	002

5. Define each Application Server to run as a `wasadmin` process. Substitute the name of each server for `server1`.

   Click **Servers > Application Servers > server1 > Process Definition > Process Execution** and change these values:

Property	Value
Run As User	wasadmin
Run As Group	mqm
UMASK	002

6. If running the WebSphere JMS provider, define the `jmsserver` process to run as a `wasadmin` process.

   Click **JMS Servers > jmsserver** *(for the node)* **> Process Definition > Process Execution** and change these values:

Property	Value
Run As User	wasadmin

Property	Value
Run As Group	mqm
UMASK	002

7. Save and synchronize.
8. Stop all servers, including the server1 and jmsserver servers.

   Use the **stopserver** command:
   ```
   stopserver server1
   stopserver jmsserver
   ```
9. Stop the node.

   Use the **stopnode** command:
   ```
   stopnode
   ```
10. If running the WebSphere JMS provider, delete the default queue manager for the Application Server.

    From the `install_root/bin` directory, run the **deletemq** command as root:
    ```
    deletemq.sh wascell wasnode jmsserver
    ```
11. If running the WebSphere JMS provider, create the WebSphere JMS provider queue manager and broker for the Application Server.

    Run the **createmq** command as `wasadmin`:
    ```
    createmq.sh /opt/WebSphere/AppServer wascell wasnode jmsserver
    ```
12. As root, use operating system tools to change file permissions:
    ```
    chgrp mqm /opt/WebSphere
    chgrp mqm /opt/WebSphere/AppServer
    chgrp -R mqm /opt/WebSphere/AppServer/config
    chgrp -R mqm /opt/WebSphere/AppServer/logs
    chgrp -R mqm /opt/WebSphere/AppServer/wstemp
    chgrp -R mqm /opt/WebSphere/AppServer/installedApps
    chgrp -R mqm /opt/WebSphere/AppServer/temp
    chgrp -R mqm /opt/WebSphere/AppServer/tranlog
    chgrp -R mqm /opt/WebSphere/AppServer/cloudscape50
    chgrp -R mqm /opt/WebSphere/AppServer/cloudscape51
    chgrp -R mqm /opt/WebSphere/AppServer/bin/DefaultDB
    chmod g+w /opt/WebSphere
    chmod g+w /opt/WebSphere/AppServer
    chmod -R g+w  /opt/WebSphere/AppServer/config
    chmod -R g+w /opt/WebSphere/AppServer/logs
    chmod -R g+w /opt/WebSphere/AppServer/wstemp
    chmod -R g+w /opt/WebSphere/AppServer/installedApps
    chmod -R g+w /opt/WebSphere/AppServer/temp
    chmod -R g+w /opt/WebSphere/AppServer/tranlog
    chmod -R g+w /opt/WebSphere/AppServer/cloudscape50
    chmod -R g+w /opt/WebSphere/AppServer/cloudscape51
    chmod -R g+w /opt/WebSphere/AppServer/bin/DefaultDB
    ```
13. Log in as `wasadmin`.
14. From wasadmin, run the startNode command to start the nodeagent process:
    ```
    startnode
    ```
15. From wasadmin, run the startserver command to start the jmsserver and all Application Servers:
    ```
    startserver jmsserver
    startserver server1
    ```
16. If running the WebSphere JMS provider, verify that the MQ queue is running:

    Run the **dspmq** command:
    ```
    dspmq
    ```

The name of the queue is `WAS_wasnode_jmsserver`.

Results

You can start an Application Server, the jmsserver, and the nodeagent from a non-root user.

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. If messages indicate that a server has stopped running, use the Application Servers page to try restarting the server. If a cluster of servers has stopped running, use the (Server Cluster page) to try (restarting the cluster). If the status of an application server is *Unavailable*, the node agent is not running in that node and you must restart the node agent before you can start the server.
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 an Application Server stops a server process based on the process definition settings in the current Application Server configuration.

Steps for this task

1. **(Optional)** Use the stopServer command to stop an Application Server from the command line.
2. **(Optional)** Use the administrative console to stop an Application Server:
   a. Click **Servers > Application Servers** from the administrative console navigation tree to go to the Application Server page.
   b. Place a checkmark in the check box beside the Application Server that you want stopped and click **Stop**.
   c. Confirm that you want to stop the Application Server.
   d. View the **Status** value and any messages or logs to see whether the Application Server stops.

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

The internal HTTP transport allows HTTP requests to be routed to the application server directly or indirectly through a Web server plug-in. By default, the internal HTTP transport listens for HTTP requests on port 9080 and for HTTPS requests on port 9443.

For example, use the URL `http://localhost:9080/snoop` to send requests to the snoop servlet on the local machine over HTTP and `https://localhost:9443/snoop` to send requests to the snoop servlet on the local machine over HTTPS.

At times, you might be able to configure the internal transport to use ports other than 9080 and 9443. 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.

What to do next

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. Specify a port number between 1 and 65535. The port number must be unique for each application server on a given machine.

Data type	Integer
Range	1 to 65535

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: Setting custom properties for an HTTP transport

WebSphere Application Server has several transport properties that are not shown in the settings page for an HTTP transport. They are as follows:

ConnectionIOTimeout
Specifies the maximum number of seconds to wait when trying to read or process data during a request. Data type: Integer.

The default value is five seconds. This value determines how long to wait while receiving two subsequent data packets for the same HTTP request. For example, using the default ConnectionIOTimeout setting of five seconds, if an HTTP client sends two data packets spaced six seconds apart, the timeout will fire, and a java.io.InterruptedIOException is raised. This will terminate the HTTP request and it will not be read.. The HTTP client will have to reissue the request.

ConnectionKeepAliveTimeout
Specifies the maximum number of seconds to wait for the next request on a keep alive connection. Data type: Integer.

MaxKeepAliveConnections
Specifies the maximum number of concurrent keep alive (persistent) connections across all HTTP transports. To make a particular transport close connections after a request, you can set MaxKeepAliveConnections to `0` (zero) or you can set KeepAliveEnabled to `false` on that transport.

The Web server plug-in keeps connections open to the application server as long as it can. However, if the value of this property is too small, performance is negatively impacted because the plug-in has to open a new connection for each request instead of sending multiple requests through one connection. The application server might not accept a new connection under a heavy load if there are too many sockets in TIME_WAIT state. If all client requests are going through the Web server plug-in and there are many TIME_WAIT state sockets for port 9080, the application server is closing connections prematurely, which decreases performance. The application server closes the connection from the plug-in, or from any client, for any of the following reasons:

- The client request was an HTTP 1.0 request when the Web server plug-in always sends HTTP 1.1 requests.
- The maximum number of concurrent keep-alives was reached. A keep-alive must be obtained only once for the life of a connection, that is, after the first request is completed, but before the second request can be read.
- The maximum number of requests for a connection was reached, preventing denial of service attacks in which a client tries to hold on to a keep-alive connection forever.
- A time out occurred while waiting to read the next request or to read the remainder of the current request.

Data type	Integer
Default	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.This parameter can help prevent denial of service attacks when a client tries to hold on to a keep-alive connection. The Web server plug-in keeps connections open to the application server as long as it can, providing optimum performance.

Data type	Integer
Default	100

KeepAliveEnabled
Specifies whether to keep connections alive or not. Data type: Boolean. Default is `true`.

You can set these properties on either the Web Container or HTTP Transport **Custom Properties** pages. When set on the Web container Custom Properties page, all transports inherit the properties. Setting the same properties on a transport overrides like settings defined for a Web container.

To specify values for these custom properties for a specific transport on the HTTP Transport **Custom Properties** page:

Steps for this task

1. Access the settings page for transport properties:
   a. In the console navigation tree, click **Servers > Application Servers >** *server_name* **> Web Container > HTTP Transport**
   b. Click on the **HOST** whose properties you want to set.

   c. Under **Additional Properties** select **Custom Properties**.
   d. On the Custom Properties page, click **New**.
2. On the settings page for a new property, enter the name of the transport property and the value to which you want it set. For example, if you want the transport to wait a maximum of 60 seconds when trying to read or write data during a request, enter `ConnectionIOTimeout` for name and `60` for value. Then click **OK**.
3. Click **Save** on the console taskbar and save the changes to the configuration.
4. Restart the server.
5. Regenerate the Web server plug-in.

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.

Configuring access logging for internal Web server HTTP transport

In order to debug potential problems with using HTTP transport as an internal Web server, you may use the following access logging capabilities.

Steps for this task

1. To turn access logging on, add the following property to the transport section of the `server.xml` file, and set the value to **false**:
   ```
   Property name: AccessLogDisable
   Values: True/False
   Default: Access log is disabled by default
   Scope:  Virtual/Global
   ```
2. To specify your own access log file, add the following property to the transport section of the `server.xml` file:
   ```
   Property name: AccessLog
   Value: <filename>
   Default Value: logs/<server instance>/http_access.log
   ```

   The default access log file is `logs/<`*`server_instance`*`>/http_access.log`. Access log entries should have the format: `<hostname or IP> <user agent> [<local time> -<status code>] <thread id> <http request> <status code> <bytecount>`.
3. To turn access logging off, add the following property to the transport section of the `server.xml` file and set the value to **true**:
   ```
   Property name: AccessLogDisable
   Values: True/False
   Default: Access log is disabled by default
   Scope:  Virtual/Global
   ```
4. Restart the server.

If you have enabled access logging, there will be an access log in the location you specified.

Parent topic: (trun_plugin_transport.html)

Related tasks (Configuring error logging for internal Web server HTTP transport)

Configuring error logging for internal Web server HTTP transport

In order to debug potential problems with using HTTP transport as an internal Web server, you may use the following error logging capabilities.

Steps for this task

1. To turn error logging on, add the following property to the transport section of the `server.xml` file and set the value to **false**:
   ```
   Property name: ErrorLogDisable
   Value: True/False
   Default: Error log is disabled by default
   Scope: Virtual/Global
   ```
2. To specify your own error log file, add the following property to the transport section of the `server.xml` file
   ```
   Property name: ErrorLog
   Value: <filename>
   Default: logs/<server instance>/http.log
   ```

   The error log property is used to specify where to place the error log. For example:<properties xmi:id="WebContainer_Property_6" name="ErrorLog" value="logs/<server instance>/http.log"/>

   Note: The error log should appear in each instance of the server.
3. Add the LogLevel property to the transport section of the `server.xml` file to specify the level of messages to log in error log file.
   ```
   Property name: LogLevel
   Value:  <level> (Levels include: debug, info, warn, error, crit)
   Default:  warn (warn includes error and crit; debug includes all levels)
   Scope:  Virtual/Global
   ```

   Log levels specify what type of message appears in the error log. The warn, error, and crit messages are logged by default.
4. To disable error logging, add the ErrorLogDisable property to the transport section of the `server.xml` file and set the value to **true**:
   ```
   Property name: ErrorLogDisable
   Value: True/False
   Default: Error log is disabled by default
   Scope:  Virtual/Global
   ```
5. Restart the server.

If you have enabled error logging and encounter an error, there should be an error log message in the error log file you specified.

Parent topic: (trun_plugin_transport.html)

Related tasks
(Configuring access logging for internal Web server HTTP transport)

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 com.ibm.websphere.runtime.CustomService interface.

   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 defintions 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 defintion 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 defintion, 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**.

Start Command

Specifies the platform-specific command to launch the server process.

Control process

Data type	String
Format	START *control JCL procedure name*
Example	START BBO5ACR

Servant process

For the servant process, the value on the start command specifies the procedure name that Workload Manger (WLM) uses to start the servant process. This value is used only if the WLM Dynamic Application Environment feature is installed.

Data type	String
Format	START *servant JCL procedure name*
Example	START BBO5ASR

Start Command Args

Specifies any additional arguments required by the start command.

Control process

Data type	String
Format	JOBNAME=*server short name*,ENV=*cell short name.node short name.server short name*
Example	JOBNAME=BBOS001,ENV=SY1.SY1.BBOS001

Servant process

Data type	String
Format	JOBNAME=*server short name*S,ENV=*cell short name.node short name.server short name*
Example	JOBNAME=BBOS001S,ENV=SY1.SY1.BBOS001

Stop Command

Specifies the platform-specific command to stop the server process

Specify two commands in the field, one for the Stop command and one for the Immediate Stop (CANCEL) command.

Data type	String
Format	STOP *server short name*;CANCEL *server short name*
Example	STOP BBOS001;CANCEL BBOS001

Stop Command Args

Specifies any additional arguments required by the stop command.

Specify arguments for the Stop command and the Immediate Stop (CANCEL) command.

Data type	String
Format	*stop command arg string*;*immediate stop command arg string*
Example	;ARMRESTART **Note:** In this example, Stop has no arguments. Immediate Stop has the argument ARMRESTART. A semicolon precedes ARMRESTART.

Terminate Command

Specifies the platform-specific command to terminate the server process

Data type	String
Format	FORCE *server short name*
Example	FORCE BBOS001

Terminate Command Args

Specifies any additional arguments required by the terminate command.

The default is an empty string.

Data type	String
Format	*terminate command arg string*
Example	ARMRESTART

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 admininistor 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 is directed. The file name can include a symbolic path name defined in the variable entries.

Use the field on the configuration tab to specify the file name. Use the field on the runtime tab to select a file for viewing. View the file by clicking **View**.

Direct server output to the administrative console or to the process that launched the server, by either deleting the file name or specifying `console` on the configuration tab.

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.

Use the field on the configuration tab to specify the file name. Use the field on the runtime tab to select a file for viewing. View the file by clicking **View**.

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.

Increasing the minimum heap size can improve startup. The number of garbage collection occurrences are reduced and a 10% gain in performance is realized.

In general, increasing the size of the Java heap improves throughput until the heap no longer resides in physical memory. After the heap begins swapping to disk, Java performance suffers drastically.

Data type	Integer
Default	64 for OS/400, 50 for all other platforms

Maximum Heap Size

Specifies the maximum heap size available to the JVM code, in megabytes.

Increasing the heap size can improve startup. The number of garbage collection occurrences are reduced and a 10% gain in performance is realized.

In general, increasing the size of the Java heap improves throughput until the heap no longer resides in physical memory. After the heap begins swapping to disk, Java performance suffers drastically. Therefore, set the maximum heap size low enough to contain the heap within physical memory.

Data type	Integer
Default	0 for OS/400, 256 for all other platforms. Keep the value low enough to avoid paging or swapping-out-memory-to-disk.

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.

The following are optional command line arguments that you can use by entering them into the **General JVM Arguments** field:

- **-Xquickstart:** You can use this value for initial compilation at a lower optimization level than in default mode, and later, depending on sampling results, you can recompile to the level of the initial compile in default mode. Use quickstart for applications where early moderate speed is more important than longrun throughput. In some debug scenarios, test harnesses and short-running tools, it is possible to realize startup time gains between 15-20%. **-DCOPT_NQREACHDEF** can improve startup by an additional 15%.
- **-Xverify:none:** When using this value, the class verification stage is skipped during class loading . By using **-Xverify:none** with the just in time (JIT) compiler enabled, startup time is improved by 10-15%.
- **-Xnoclassgc:** You can use this value to disable class garbage collection, making class reuse more available, and slightly improving performance. Class garbage collection is enabled by default, but it is recommended that you enable it. You can monitor garbage collection using the verbose:gc configuration setting because its output includes class garbage collection statistics.
- **-Xgcthreads:** You can use several garbage collection threads at one time, also known as *parallel garbage collection*. When entering this value in the **Generic JVM Arguments** field, also enter the number of processors that your machine has, for example, `-Xgcthreads=` *number_of_processors*. It is recommended that you use parallel garbage collection if your machine has more than one processor. This argument applies only to the IBM Developer Kit.
- **-Xnocompactgc:** This value disables heap compaction which is the most expensive garbage collection operation. Avoid compaction in IBM Developer Kit 1.3. If you disable heap compaction, you eliminate all associated overhead. When entering this value in the **Generic JVM Arguments** field, also enter the number of processors that your machine has, for example, `-Xnocompactgc=` *number_of_processors*.
- **-Xinitsh:** You can use this value to set the initial heap size where class objects are stored. The method definitions and static fields are also stored with the class objects. Although the system heap size has no upper bound, set the initial size so that you do not incur the cost of expanding the system heap size, which involves calls to the operating system memory manager. You can compute a good initial system heap size by knowing the number of classes loaded in the WebSphere product, which is about 8,000 classes, and their average size. Having knowledge of the application helps you include them in the calculation.
- **-Xmc:** The thread local heap size is a portion of the heap that is allocated exclusively for a thread. Because of the thread local heap size, the thread does not need to lock the entire heap when allocating objects. However, when the thread local heap is full, object allocation is done from the heap that needs synchronization. A good local cache size is critical to performance and requires knowledge of the application and its objects.
- **-Xml:** You can use this value to set the limit of an object size to allocate from the local cache. Objects that exceed the limit size need allocating in the regular heap. Allocate objects from the local cache as much as possible or the local cache depletes because it does not grow dynamically. If you know some objects are going to be very large, allocate them from the regular heap.

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.

If you disable the JIT compiler, throughput decreases noticeably. Therefore, for performance reasons, keep JIT enabled.

Data type	Boolean
Default	false (JIT enabled)
Recommended	JIT enabled

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.

Java memory tuning tips

Enterprise applications written in the Java language involve complex object relationships and utilize large numbers of objects. Although, the Java language automatically manages memory associated with object life cycles, understanding the application usage patterns for objects is important. In particular, verify the following:

- The application is not over-utilizing objects
- The application is not leaking objects
- The Java heap parameters are set properly to handle a given object usage pattern

Understanding the effect of garbage collection is necessary to apply these management techniques.

The garbage collection bottleneck

Examining Java garbage collection gives insight to how the application is utilizing memory. Garbage collection is a Java strength. By taking the burden of memory management away from the application writer, Java applications are more robust than applications written in languages that do not provide garbage collection. This robustness applies as long as the application is not abusing objects. Garbage collection normally consumes from 5% to 20% of total execution time of a properly functioning application. If not managed, garbage collection is one of the biggest bottlenecks for an application, especially when running on symmetric multiprocessing (SMP) server machines. The Java virtual machine (JVM) uses a parallel garbage collector to fully exploit an SMP during most garbage collection cycles where the Sun HotSpot 1.3.1 JVM has a single-threaded garbage collector. For more information about garbage collection in a Solaris operating environment see Performance: Resources for learning.

The garbage collection gauge

You can use garbage collection to evaluate application performance health. By monitoring garbage collection during the execution of a fixed workload, you gain insight as to whether the application is over-utilizing objects. Garbage collection can even detect the presence of memory leaks.

You can monitor garbage collection statistics using object statistics in the Tivoli Performance Viewer, or using the **verbose:gc** JVM configuration setting. The

verbose:gc format is not standardized between different JVMs or release levels. For a description of the IBM verbose:gc output and more information about the IBM garbage collector, see Performance: Resources for learning.

For this type of investigation, set the minimum and maximum heap sizes to the same value. Choose a representative, repetitive workload that matches production usage as closely as possible, user errors included.

To ensure meaningful statistics, run the fixed workload until the application state is steady. It usually takes several minutes to reach a steady state.

Detecting over-utilization of objects

You can use the Tivoli Performance Viewer to check if the application is overusing objects, by observing the counters for the JVM runtime. You have to set the **-XrunpmiJvmpiProfiler** command line option, as well as the JVM module maximum level in order to enable the Java virtual machine profiler interface (JVMPI) counters. The best result for the average time between garbage collections is at least 5-6 times the average duration of a single garbage collection. If you do not achieve this number, the application is spending more than 15% of its time in garbage collection.

If the information indicates a garbage collection bottleneck, there are two ways to clear the bottleneck. The most cost-effective way to optimize the application is to implement object caches and pools. Use a Java profiler to determine which objects to target. If you can not optimize the application, adding memory, processors and clones might help. Additional memory allows each clone to maintain a reasonable heap size. Additional processors allow the clones to run in parallel.

Detecting memory leaks

Memory leaks in the Java language are a dangerous contributor to garbage collection bottlenecks. Memory leaks are more damaging than memory overuse, because a memory leak ultimately leads to system instability. Over time, garbage collection occurs more frequently until the heap is exhausted and the Java code fails with a fatal `Out of Memory` exception. Memory leaks occur when an unused object has references that are never freed. Memory leaks most commonly occur in collection classes, such as Hashtable because the table always has a reference to the object, even after real references are deleted.

High workload often causes applications to crash immediately after deployment in the production environment. This is especially true for leaking applications where the high workload accelerates the magnification of the leakage and a memory allocation failure occurs.

The goal of memory leak testing is to magnify numbers. Memory leaks are measured in terms of the amount of bytes or kilobytes that cannot be garbage collected. The delicate task is to differentiate these amounts between expected sizes of useful and unusable memory. This task is achieved more easily if the numbers are magnified, resulting in larger gaps and easier identification of inconsistencies. The following list contains important conclusions about memory leaks:

- **Long-running test**

  Memory leak problems can manifest only after a period of time, therefore, memory leaks are found easily during long-running tests. Short running tests can lead to false alarms. It is sometimes difficult to know when a memory leak is occurring in the Java language, especially when memory usage has seemingly increased either abruptly or monotonically in a given period of time. The reason it is hard to detect a memory leak is that these kinds of increases can be valid or might be the intention of the developer. You can learn how to differentiate the delayed use of objects from completely unused objects by running applications for a longer period of time. Long-running application testing gives you higher confidence for whether the delayed use of objects is actually occurring.

- **Repetitive test**

  In many cases, memory leak problems occur by successive repetitions of the same test case. The goal of memory leak testing is to establish a big gap between unusable memory and used memory in terms of their relative sizes. By repeating the same scenario over and over again, the gap is multiplied in a very progressive way. This testing helps if the number of leaks caused by the execution of a test case is so minimal that it is hardly noticeable in one run.

  You can use repetitive tests at the system level or module level. The advantage with modular testing is better control. When a module is designed to keep the private module without creating external side effects such as memory usage, testing for memory leaks is easier. First, the memory usage before running the module is recorded. Then, a fixed set of test cases are run repeatedly. At the end of the test run, the current memory usage is recorded and checked for significant changes. Remember, garbage collection must be suggested when recording the actual memory usage by inserting System.gc() in the module where you want garbage collection to occur, or using a profiling tool, to force the event to occur.
- **Concurrency test**

  Some memory leak problems can occur only when there are several threads running in the application. Unfortunately, synchronization points are very susceptible to memory leaks because of the added complication in the program logic. Careless programming can lead to kept or unreleased references. The incident of memory leaks is often facilitated or accelerated by increased concurrency in the system. The most common way to increase concurrency is to increase the number of clients in the test driver.

  Consider the following points when choosing which test cases to use for memory leak testing:
  - A good test case exercises areas of the application where objects are created. Most of the time, knowledge of the application is required. A description of the scenario can suggest creation of data spaces, such as adding a new record, creating an HTTP session, performing a transaction and searching a record.
  - Look at areas where collections of objects are used. Typically, memory leaks are composed of objects within the same class. Also, collection classes such as Vector and Hashtable are common places where references to objects are implicitly stored by calling corresponding insertion methods. For example, the get method of a Hashtable object does not remove its reference to the retrieved object.

Tivoli Performance Viewer can help find memory leaks. For best results, repeat experiments with increasing duration, like 1000, 2000, and 4000-page requests. The Tivoli Performance Viewer graph of used memory should have a sawtooth shape. Each drop on the graph corresponds to a garbage collection. There is a memory leak if one of the following occurs:

- The amount of memory used immediately after each garbage collection increases significantly. The sawtooth pattern looks more like a staircase.
- The sawtooth pattern has an irregular shape.

Also, look at the difference between the number of objects allocated and the number of objects freed. If the gap between the two increases over time, there is a memory leak.

Heap consumption indicating a possible leak during a heavy workload (the application server is consistently near 100% CPU utilization), yet appearing to recover during a subsequent lighter or near-idle workload, is an indication of heap fragmentation. Heap fragmentation can occur when the JVM can free sufficient objects to satisfy memory allocation requests during garbage collection cycles, but the JVM does not have the time to compact small free memory areas in the heap to larger contiguous spaces.

Another form of heap fragmentation occurs when small objects (less than 512 bytes) are freed. The objects are freed, but the storage is not recovered, resulting in memory fragmentation until a heap compaction has been run.

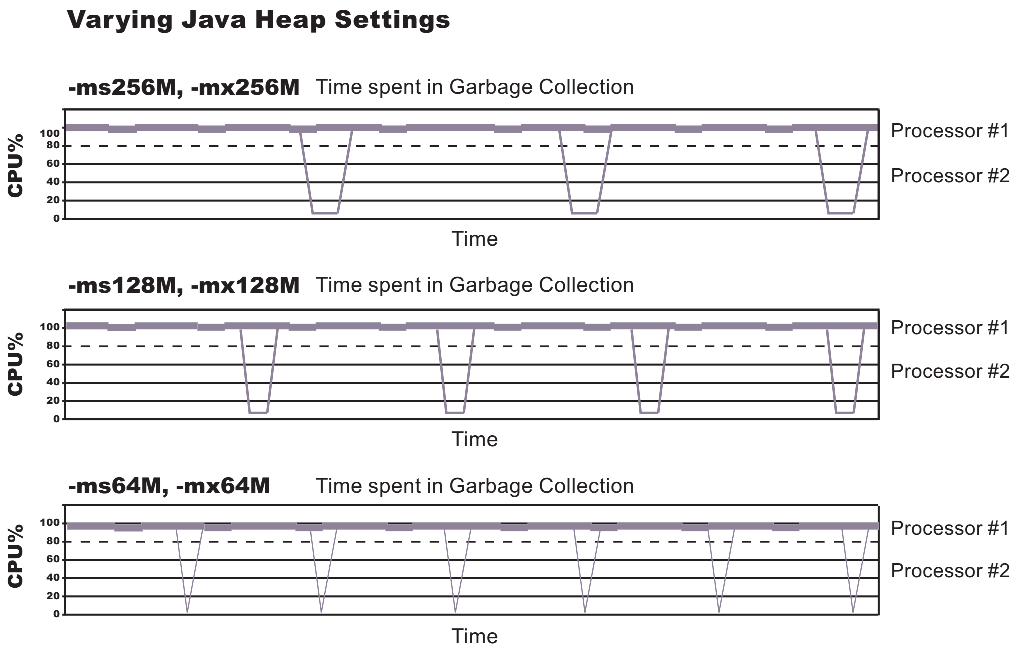
To avoid heap fragmentation, turn on the -Xcompactgc flag in the JVM advanced settings command line arguments. The -Xcompactgc function verifies that each garbage collection cycle eliminates fragmentation. However, compaction is a relatively expensive operation. See Heap compaction (-Xnocompactgc) for more information.

Java heap parameters

The Java heap parameters also influence the behavior of garbage collection. Increasing the heap size supports more object creation. Because a large heap takes longer to fill, the application runs longer before a garbage collection occurs. However, a larger heap also takes longer to compact and causes garbage collection to take longer. SeeHeap compaction for more information.

For performance analysis, the initial and maximum heap sizes should be equal.

When tuning a production system where the working set size of the Java application is not understood, a good starting value for the initial heap size is 25% of the maximum heap size. The JVM then tries to adapt the size of the heap to the working set size of the application.



The illustration represents three CPU profiles, each running a fixed workload with varying Java heap settings. In the middle profile, the initial and maximum heap sizes are set to 128MB. Four garbage collections occur. The total time in garbage collection is about 15% of the total run. When the heap parameters are doubled to 256MB, as in the top profile, the length of the work time increases between garbage collections. Only three garbage collections occur, but the length of each garbage collection is also increased. In the third profile, the heap size is reduced to 64MB and exhibits the opposite effect. With a smaller heap size, both the time between garbage collections and the time for each garbage collection are shorter. For all three configurations, the total time in garbage collection is approximately 15%. This example illustrates an important concept about the Java heap and its relationship to object utilization. There is always a cost for garbage collection in Java applications.

Run a series of test experiments that vary the Java heap settings. For example, run experiments with 128MB, 192MB, 256MB, and 320MB. During each experiment, monitor the total memory usage. If you expand the heap too aggressively, paging can occur. Use the **vmstat** command or the Windows NT or Windows 2000 Performance Monitor to check for paging. If paging occurs, reduce the size of the heap or add more memory to the system. When all the runs are finished, compare the following statistics:

- Number of garbage collection calls
- Average duration of a single garbage collection call
- Ratio between the length of a single garbage collection call and the average time between calls

If the application is not over-utilizing objects and has no memory leaks, the state of steady memory utilization is reached. Garbage collection also occurs less frequently and for short duration.

If the heap free space settles at 85% or more, consider decreasing the maximum heap size values because the application server and the application are under-utilizing the memory allocated for heap.

For more information about garbage collection see Performance: Resources for learning.

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 instructions and examples
- Programming specifications
- Administration

Programming instructions and examples

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

Programming specifications

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

Administration

- **Listing of all IBM WebSphere Application Server Redbooks** (http://publib-b.boulder.ibm.com/Redbooks.nsf/Portals/WebSphere)

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. **(Optional)** Use ORB custom property settings to enable and configure the Logical Pool Distribution (LPD) mechanism to improve performance on requests that have shorter execution times.

   LPD provides a mechanism allow shorter execution time requests greater access to execution threads. The LPD defaults are usually sufficient. However, when activating the LPD, it may be necessary to increase the ORB maximum thread pool size by 33 to 200 percent. Testing may be required to determine the best overall values for the ORB maximum thread pool size. Results will vary among systems and applications.

   For more information about LPD, see ″Logical Pool Distribution.″
4. 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.

Logical Pool Distribution (LPD)

The Logical Pool Distribution (LPD) thread pool mechanism implements a strategy for improving the performance of requests that have shorter execution times.

The need for LPD is indicated by a mixture of EJB requests where the execution times vary across the request types, and the ORB thread pool must be constrained for performance reasons. In this case, longer execution time requests may tend to elongate the response times for shorter execution time requests by denying them adequate access to threads in the thread pool. LPD provides a mechanism allow shorter execution time requests greater access to execution threads.

LPD divides the Object Request Broker (ORB) thread pool into logical pools, as configured by the administrator using ORB custom properties starting with com.ibm.websphere.threadpool.strategy.*. The size of each pool is a percentage of the maximum number of ORB threads. The sum of the logical pool percentages should equal 100.

When LPD is active, incoming ORB requests are vectored to a pool based on historical execution time history for the request type. The request type is determined by the method which is qualified internally as unique across components. The LPD mechanism adjusts pool targets at runtime to optimize the distribution of requests across logical pools.

After it is enabled, the LPD mechanism can be tuned. Tuning exercises should be driven by response time and throughput measurements, as well as statistics produced by the LPD mechanism.

Object Request Broker tuning guidelines

The following options exist for improving the performance of the Object Request Broker (ORB). Tuning results will vary among systems and applications.

- **Logical Pool Distribution (LPD) mechanism**

  If you suspect that requests with longer execution times are elongating the response times for shorter execution time requests by denying them adequate access to threads in the thread pool, LPD provides a mechanism to allow the shorter requests greater access to execution threads.

  For more information, see ″Logical Pool Distribution (LPD).″
- **ORB timeout**

  If Web clients that access Java applications running in the product environment are consistently experiencing problems with their requests, and the problem cannot be traced to other sources and addressed through other solutions, consider setting an ORB 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 wait before timing out, setting an ORB time-out requires 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 threshhold 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

- **com.ibm.CORBA.numJNIReaders system property**

  You can 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.

- **Determining the ORB message size**

  The ORB breaks apart messages into fragments to send over the ORB connection. You can configure this fragment size through the com.ibm.CORBA.FragmentSize parameter.

  To determine the size of the messages being transferred over the ORB and the number of fragments required to do so, you must first enable ORB tracing in the ORB Properties page in the webui and then enable ORBRas tracing from the logging and tracing page in the webui. You'll probably also want to bump up the trace file sizes as this can generate a lot of data. Restart the server and run at least one iteration (preferably several) of the case you wish to measure.

  Then look at the traceable file and do a search for ″Fragment to follow: Yes″. This indicates that the ORB transmitted a fragment, but it still has at least one remaining fragment to send before the entire message has arrived. If No is indicated instead of Yes, this means that that particular fragment is the last in the entire message. It may also be the first if the message fit entirely into one fragment.

If you go to the spot where ″Fragment to follow: Yes″ is located, you will find a block that looks similar to this:

Fragment to follow:	Yes
Message size:	4988 (0x137C)
—	
Request ID:	1411

This indicates that the amount of data in the fragment is 4988 bytes and the Request ID is 1411. Then if you do a search for all occurrences of ″Request ID: 1411″, you will see the number of fragments used to send that particular message. If you add all the associated message sizes, you will have the total size of the message that's being send through the ORB.

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**.

Several settings are available for controlling internal Object Request Broker (ORB) processing. You can use these settings to improve application performance in the case of applications containing enterprise beans. You can make changes to these settings for the default server or any application server configured in the administrative domain.

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. If there are many simultaneous clients connecting to the server-side ORB, this parameter can be increased to support the heavy load up to 1000 clients.

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

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 (`false`)

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

Specifies how the ORB passes parameters. If enabled, the ORB passes parameters by reference instead of by value, which avoids making an object copy. If you do not enable pass by reference, the parameters are copied to the stack before every remote method call is made, which can be expensive.

If the EJB client and the EJB server are installed in the same WebSphere Application Server instance, and the client and server use remote interfaces, enabling Pass by reference can improve performance up to 50%. Pass by reference helps performance only where non-primitive object types are passed as parameters. Therefore, int and floats are always copied, regardless of the call model.

Enable this property with caution, because unexpected behavior can occur. If an object reference is modified by the remote method, the caller might change.

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)

The use of this option for enterprise beans with remote interfaces violates the EJB Specification, Version 2.0 (see section 5.4). Object references passed to EJB methods or to EJB home methods are not copied and can be subject to corruption.

Consider the following example:

```
Iterator iterator = collection.iterator();
MyPrimaryKey pk = new MyPrimaryKey();
while (iterator.hasNext()) {
   pk.id = (String) iterator.next();
   MyEJB myEJB = myEJBHome.findByPrimaryKey(pk);
}
```

In this example, a reference to the same MyPrimaryKey object passes into WebSphere Application Server with a different ID value each time. Running this code with Pass by reference `enabled` causes a problem within the application server because multiple enterprise beans are referencing the same MyPrimaryKey object. To avoid this problem, set the system property com.ibm.websphere.ejbcontainer.allowPrimaryKeyMutation to `true` when Pass by reference is enabled. Setting Pass by reference to `true` causes the EJB container to make a local copy of the PrimaryKey object. As a result, however, a small portion of the performance advantage of setting Pass by reference is lost.

As a general rule, any application code that passes an object reference as a parameter to an enterprise bean method or EJB home method must be scrutinized to determine if passing that object reference results in loss of data integrity or other problems.

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 the administrative console page, click **Servers > Application Servers >** *serverName* **> ORB Service > Custom Properties**.

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

The page already might include Secure Sockets 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. 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.″

Default	2809

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. This should also be set on the client side with a -D system property if using a stand-alone java application.

Consider adjusting this when the amount of data that is sent over IIOP exceeds 1k. Definitly adjust this if thread dumps show most client side threads stuck in a wait coming from writeLock. Sometimes a low amount of fragmentation is ok, so the parameter should be set such that most messages have few to no fragments.

Units	Bytes.
Default	1024
Range	From 64 to largest value of 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.

Default	Next available system-assigned port number
Range	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

Corresponds to the length of the TCP/IP stack listen queue and prevents WebSphere Application Server from rejecting requests when there is not space in the listen queue. If there are several simultaneous clients connecting to the server-side ORB, you can increase this parameter to support up to 1000 clients.

Default	50
Range	From 50 to the largest value of Java int type

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.

com.ibm.websphere.threadpool.strategy.implementation

If set to com.ibm.ws.threadpool.strategy.LogicalPoolDistribution, this enables the Logical Pool Distribution (LPD) thread pool strategy the next time you start the application server.

Some requests have shorter execution times than others. LPD is a mechanism for allowing these shorter requests greater access to execution threads. For more information, see ″Logical Pool Distribution.″

com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.calcinterval

Specifies how often the Logical Pool Distribution (LPD) mechanism will readjust the pool execution target times. It cannot be turned off once this support is installed.

LPD must be enabled (see com.ibm.websphere.threadpool.strategy.implementation).

Data type	Integer
Units	Milliseconds
Default	30
Range	20,000 minimum

com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.lruinterval

Specifies how long the Logical Pool Distribution internal data is kept for inactive requests. The mechanism tracks several statistics for each request type received. Consider removing requests that have not been active for a while.

LPD must be enabled (see com.ibm.websphere.threadpool.strategy.implementation).

Data type	Integer
Units	Milliseconds
Default	300,000 (5 minutes)
Range	60,000 (1 minute) minimum

com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.outqueues

Specifies how many pools are created and how many threads are allocated to each pool in the Logical Pool Distribution mechanism

The ORB parameter for max threads controls the total number of threads. The outqueues parameter is specified as a comma separated list of percentages that should add up to 100. For example, the list 25,25,25,25 will set up 4 pools, each allocated 25% of the available ORB thread pool. The pools are indexed left to right from 0 to n-1. Each outqueue is dynamically assigned a target execution time by the calculation mechanism. Target execution times are assigned to outqueues in increasing order so pool 0 gets the requests with the least execution time and pool n-1 gets requests with the highest execution times.

LPD must be enabled (see com.ibm.websphere.threadpool.strategy.implementation).

Data type	Integers in comma separated list
Default	25,25,25,25
Range	Percentages in list must total 100 percent

com.ibm.websphere.threadpool.strategy. LogicalPoolDistribution.statsinterval

If active, statistics will be dumped to stdout after this interval expires, but only if requests have been processed. This keeps the mechanism from filling the log files with redundant information. These stats are beneficial for tuning the Logical Pool Distribution mechanism.

LPD must be enabled (see com.ibm.websphere.threadpool.strategy.implementation).

Data type	Integer
Units	Milliseconds
Default	0 (meaning Off)
Range	30,000 (30 seconds) minimum

com.ibm.websphere.threadpool.strategy. LogicalPoolDistribution.workqueue

Specifies the size of a new queue where incoming requests wait for dispatching. Pertains to the Logical Pool Distribution mechanism.

LPD must be enabled (see com.ibm.websphere.threadpool.strategy.implementation).

Data type	Integer
Default	96
Range	10 minimum

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:O=1:CT
6.  Local Port:   1243 (0x4DB)
7.  Local IP:     jdoe.austin.ibm.com/192.168.1.101
8.  Remote Port:  1242 (0x4DA)
9.  Remote IP:    jdoe.austin.ibm.com/192.168.1.101
10.  GIOP Version:  1.2
11. Byte order:   big endian
12. Fragment to follow: No
13. Message size:  268 (0x10C)
--
15. Request ID:       5
16. Response Flag:    WITH_TARGET
17. Target Address:    0
```

```
18. Object Key:        length = 24 (0x18)
                       4B4D4249 00000010 BA4D6D34 000E0008
                       00000000 00000000
21. Operation:         _get_value
22. Service Context:   length = 3 (0x3)
23. Context ID:  1229081874 (0x49424D12)
24. Context data:  length = 8 (0x8)
                       00000000 13100003
26. Context ID:  6 (0x6)
27. Context data:  length = 164 (0xA4)
                       00000000 00000028 49444C3A 6F6D672E
                       6F72672F 53656E64 696E6743 6F6E7465
                       78742F43 6F646542 6173653A 312E3000
                       00000001 00000000 00000068 00010200
                       0000000E 3139322E 3136382E 312E3130
                       310004DC 00000018 4B4D4249 00000010
                       BA4D6D69 000E0008 00000000 00000000
                       00000002 00000001 00000018 00000000
                       00010001 00000001 00010020 00010100
                       00000000 49424D0A 00000008 00000000
                       13100003
39. Context ID:  1 (0x1)
40. Context data:  length = 12 (0xC)
                       00000000 00010001 00010100
42. Data Offset:       118


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

Sample Log Entry - GIOP Reply

```
1.  IN COMING:

3.  Reply Message
4.  Date:         April 17, 2002 10:00:47 PM CDT
5.  Thread Info:   RT=0:P=842115:O=1:com.ibm.rmi.transport.TCPTransportConnection
5a.   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 `ISO8859_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. If no codeset for wide character data is found and this property is not set, the ORB raises an exception, as specified in the CORBA specification. There is no default value set for this property. 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	

Java name	OSF registry ID	Comments
ISO8859_4	0x00010004	
ISO8859_5	0x00010005	
ISO8859_6	0x00010006	
ISO8859_7	0x00010007	
ISO8859_8	0x00010008	
ISO8859_9	0x00010009	
ISO8859_15_FDIS	0x0001000F	
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)
  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)

Chapter 4. Balancing workloads with clusters

To monitor application servers and manage the workloads of servers, use server clusters and cluster members provided by the Network Deployment product.

To assist you in understanding how to configure and use clusters for workload management, below is a scenario. In this scenario, client requests are distributed among the cluster members on a single machine. (A client refers to any servlet, Java application, or other program or component that connects the end user and the application server that is being accessed.) In more complex workload management scenarios, you can distribute cluster members to remote machines.

Steps for this task

1. Decide which application server you want to cluster.
2. Decide whether you want to configure replication domains and entries. Replication enables the sharing of data among processes and the backing up of failed processes.
3. Deploy the application onto the application server.
4. After configuring the application server and the application components exactly as you want them to be, create a cluster. The original server instance becomes a cluster member that is administered through the cluster.
5. If you did not do so when creating a cluster, create one or more cluster members of the cluster.
6. **(Optional)** If you did not do so when creating a cluster, configure a weight advisor to assist the cluster in balancing the work loads of cluster members.
7. **(Optional)** Configure a backup cluster that handles requests if the primary cluster fails.
8. Start all of the application servers by starting the cluster. Workload management automatically begins when you start the cluster members of the application server.
9. Stop the cluster.
10. Upgrade applications on clusters.
11. (Detect and handle problems with server clusters and their workloads.)
12. **(Optional)** Tune the behavior of the workload management run time. If your application is experiencing problems with timeouts or your network experiences extreme latency, change the timeout interval for the com.ibm.CORBA.RequestTimeout property. Or,if the workload management state of the client is refreshing too soon or too late, change the interval for the com.ibm.websphere.wlm.unusable.interval property.

You need to define a bootstrap host for stand-alone Java clients, which are clients located on a different machine from the application server that have no administrative server. Add the following line to the Java Virtual Machine (JVM) arguments for the client:

```
-Dcom.ibm.CORBA.BootstrapHost=machine_name
```

where *machine_name* is the name of the machine on which the administrative server is running.

Workload management (WLM)

Workload management optimizes the distribution of client processing tasks. Incoming work requests are distributed to the application servers, enterprise beans, servlets, and other objects that can most effectively process the requests. Workload management also provides failover when servers are not available, improving application availability.

Workload management provides the following benefits to WebSphere Application Server applications:

- It balances client workloads, allowing processing tasks to be distributed according to the capacities of the different machines in the system.
- It provides failover capability by redirecting client requests if one or more servers is unable to process them. This improves the availability of applications and administrative services.
- It enables systems to be scaled up to serve a higher client load than provided by the basic configuration. With clustering, additional instances of servers, servlets, and other objects can easily be added to the configuration.
- It enables servers to be transparently maintained and upgraded while applications remain available for users.
- It centralizes the administration of servers and other objects.

In the WebSphere Application Server environment, you implement workload management by using clusters, transports, and replication domains.

Techniques for managing state

Multimachine scaling techniques rely on using multiple copies of an application server; multiple consecutive requests from various clients can be serviced by different servers. If each client request is completely independent of every other client request, it does not matter whether consecutive requests are processed on the same server. However, in practice, client requests are not independent. A client often makes a request, waits for the result, then makes one or more subsequent requests that depend on the results received from the earlier requests. This sequence of operations on behalf of a client falls into two categories:

Stateless
A server processes requests based solely on information provided with each request and does not reply on information from earlier requests. In other words, the server does not need to maintain state information between requests.

Stateful
A server processes requests based on both the information provided with each request and information stored from earlier requests. In other words, the server needs to access and maintain state information generated during the processing of an earlier request.

For stateless interactions, it does not matter whether different requests are processed by different servers. However, for stateful interactions, the server that processes a request needs access to the state information necessary to service that request. Either the same server can process all requests that are associated with the same state information, or the state information can be shared by all servers that require it. In the latter case, accessing the shared state information from the same server minimizes the processing overhead associated with accessing the shared state information from multiple servers.

The load distribution facilities in WebSphere Application Server use several different techniques for maintaining state information between client requests:

- Session affinity, where the load distribution facility recognizes the existence of a client session and attempts to direct all requests within that session to the same server.
- Transaction affinity, where the load distribution facility recognizes the existence of a transaction and attempts to direct all requests within the scope of that transaction to the same server.
- Server affinity, where the load distribution facility recognizes that although multiple servers might be acceptable for a given client requests, a particular server is best suited for processing that request.
- The WebSphere Application Server Session Manager, which is part of each application server, stores client session information and takes session affinity and server affinity into account when directing client requests to the cluster members of an application server. The workload management service takes server affinity and transaction affinity into account when directing client requests among the cluster members of an application server.

Clusters

Clusters are sets of servers that are managed together and participate in workload management. The servers that are members of a cluster can be on different host machines, as opposed to the servers that are part of the same node and must be located on the same host machine.

A cell can have no clusters, one cluster, or multiple clusters.

Servers that belong to a cluster are *members* of that cluster set and must all have identical application components deployed on them. Other than the applications configured to run on them, cluster members do not have to share any other configuration data. One cluster member might be running on a huge multi-processor enterprise server system while another member of that same cluster might be running on a small laptop. The server configuration settings for each of these two cluster members are very different, except in the area of application components assigned to them. In that area of configuration, they are identical.

A *vertical cluster* has cluster members on the same node. A *horizontal cluster* has cluster members on multiple nodes.

A network dispatcher routes application access among cluster members by server-weighting, to provide better distribution control.

WebSphere Application Server can respond to increased use of an enterprise application by automatically replicating the application to additional cluster members as needed. This lets you deploy an application on a cluster instead of on a single node, without considering workload.

Creating clusters

You can manage application servers collectively using a cluster. To create a cluster, view information about clusters, or manage server members on a cluster, use the Server Cluster page.

Steps for this task

1. Go to the Server Cluster page. Click **Servers > Clusters** in the console navigation tree. The Server Cluster page lists clusters of application servers in the cell and states whether a cluster is stopped, started or unavailable.
2. Click **New** to access the Create New Cluster page.
3. Type a cluster name.
4. **(Optional)** To enable or disable node scoped routing optimization, place a checkmark in the **Prefer local enabled** check box. The default is enabled, which indicates that, if possible, EJB requests are routed to the client's node. If you enable this feature, performance is improved because client requests are sent to local EJBs.
5. **(Optional)** To enable memory-to-memory replication of HttpSession (for failover) or replication of cached data and cache invalidations with a Web Container's dynamic caching, select options supporting data replication.
6. Choose whether to create an empty cluster or to create a cluster based on an existing server.

   To create an empty cluster, do not include an existing server in this cluster.

   To create a cluster based on an existing server member, add server members to this cluster. To add a server member, choose **Select an existing server to add to this cluster** and then, from the drop-down list, select the server you want to add.
7. Click **Next**.
8. **(Optional)** Add application servers (cluster members) to the cluster. For each new cluster member, do the following:
   a. Type the name of a new application server (cluster member) to add to the cluster.
   b. Select the node on which the server will reside.
   c. Specify the server weight. The weight value controls the amount of work directed to the application server. If the weight value for the server is greater than the weight values assigned to other servers in the cluster, then the server receives a larger share of the servers' workload. The value can range from 0 to 20.
   d. Specify whether to generate a unique HTTP port.
   e. Specify whether to create a replication entry for the server. A replication entry enables memory-to-memory replication of HttpSession (for failover) or replication of cached data and cache invalidations with a Web Container's dynamic caching.
   f. Specify the server template.
   g. Click **Apply** to finish the cluster member. Repeat the above steps to define another cluster member.
9. Click **Next** and review the summary of changes.
10. Click **Finish** to complete the configuration.
11. Click **Save** on the administrative console taskbar and save your administrative configuration. As part of saving the change to the configuration, you can select **Synchronize changes with Nodes** before clicking **Save** on the Save page.
12. Before you can start the cluster, the configuration needs to be synchronized to the nodes. If you selected **Synchronize changes with Nodes** when saving your configuration in the previous step, you can ignore this step. If you are running automatic synchronization, wait until synchronization runs. Or, run manual synchronization to get the configuration files moved to the nodes. Click **System Administration > Nodes** and, on the Nodes page, select the

node and click **Synchronize** or **Full Resynchronize**. The Nodes page displays status indicating whether the node is synchronized.

13. To further configure a cluster, click on the cluster's name under **Name**. This displays the settings for the server cluster instance. Note that, unless you have clicked **Save** and saved your administrative configuration, you only see the **Configuration** and **Local Topology** tabs; to see the **Runtime** tab as well you must save your administrative configuration. Also, ensure that changes are synchronized to the nodes (step 12).

Server cluster collection

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

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

Click **New** to access the Create New Cluster page, which you use to define a new cluster.

Name

Specifies a logical name for the cluster. The name must be unique among clusters within the containing cell.

Status

Specifies whether cluster members are stopped, starting, or running.

If all cluster members are stopped, the cluster status and state is *Stopped*. After you request to start a cluster by clicking **Start** or **Ripplestart**, the cluster state briefly changes to *Starting* and each server that is a member of that cluster launches, if it is not already running. When the first member launches, the state changes to *PartialStart*. The state remains *PartialStart* until all cluster members are running, then the state changes to *Running* and the status is *Started*. Similarly, when stopping a cluster by clicking **Stop** or **ImmediateStop**, the state changes to *PartialStop* as the first member stops and changes to *Stopped* when all members are not running.

Server cluster settings

Use this page to view or change the configuration and local topology of a server cluster instance. Provided you saved your administrative configuration after creating the server cluster instance, you can also view run-time information such as the status of the server cluster instance.

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

Cluster name

Specifies a logical name for the cluster. The name must be unique among clusters within the containing cell.

Data type	String

Cluster short name

Specifies the cluster short name for this cluster.

The name is 1-8 characters, alpha-numeric or national language. It cannot start with a numeric.

Data type	String

Unique Id

Specifies the unique ID of this cluster.

The unique ID property is read only. The system automatically generates the value.

Prefer local

Specifies whether enterprise bean requests are routed to the node on which the client resides, if it is possible to do so.

Select the **Prefer Local** check box to specify routing of requests to the node on which the client resides. By default, the **Prefer Local** check box is selected, specifying routing of requests to the node.

Data type	Boolean
Default	true

wlcID

Specifies the currently registered workload controller (WLC) identifier for the cluster. This setting might not display for all configurations.

Data type	String

State

Specifies whether cluster members are stopped, starting, or running.

If all cluster members are stopped, the cluster state is *websphere.cluster.stopped*. After you request to start a cluster, the cluster state briefly changes to *websphere.cluster.starting* and each server that is a member of that cluster launches, if it is not already running. When the first member launches, the state changes to *websphere.cluster.partial.start*. The state remains *websphere.cluster.partial.start* until all cluster members are running, then the state changes to *websphere.cluster.running*. Similarly, when stopping a cluster, the state changes to *websphere.cluster.partial.stop* as the first member stops and changes to *websphere.cluster.stopped* when all members are not running.

Data type	String
Range	Valid values are websphere.cluster.starting, websphere.cluster.partial.start, websphere.cluster.running, websphere.cluster.partial.stop, or websphere.cluster.stopped.

Creating cluster members

You create a cluster member to represent an application server in a cluster. To create a cluster member, view information about cluster members, or manage members of a cluster, use the Cluster Members page.

Steps for this task

1. Go to the Cluster Members page. Click **Servers > Clusters** in the console navigation tree. Then, click a cluster in the collection of clusters and click **Cluster Members**. The Cluster Members page lists members of a cluster, states the nodes on which members reside, and states whether members are started, stopped or encountering problems.
2. Click **New** and follow the steps on the Create New Cluster Members page.
   a. Type a name for the cluster member (application server).
   b. Select the node on which the server will reside.
   c. Specify the server weight. The weight value controls the amount of work directed to the application server. If the weight value for the server is greater than the weight values assigned to other servers in the cluster, then the server receives a larger share of the servers' workload. The value can range from 0 to 100.
   d. Specify whether to generate a unique HTTP port.
   e. Specify whether to create a replication entry for the server.
   f. Specify the server template.
   g. Click **Apply** to finish the cluster member. Repeat steps 1 through 7 to define another cluster member.
   h. Click **Next**.
   i. Review the summary of information on new cluster members and click **Finish**.
3. Click **Save** on the administrative console taskbar and save your administrative configuration.
4. To examine a cluster member's settings, click on the member's name under **Member Name** on the Cluster Members page. This displays the settings page for the cluster member instance.

Cluster member collection

Use this page to view information about and manage members of an application server cluster.

To view this administrative console page, click **Servers > Clusters >** *cluster_name* **> Cluster Members**.

Member name

Specifies the name of the server in the cluster. On most platforms, the name of the server is the process name. The name must match the (object) name of the application server.

Node

Specifies the name of the node for the cluster member.

Status

Specifies whether a cluster member is running, stopped, or unavailable.

If a cluster member is stopped, its status is *Stopped*. After you request to start a cluster member by clicking **Start**, the status becomes *Started*. After you click **Stop**, its status changes to *Stopped* when it stops running.

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 cluster member.

Cluster member settings

Use this page to configure a member instance of an application server cluster.

To view this administrative console page, click **Servers > Clusters >** *cluster_name* **> Cluster Members >** *cluster_member_name*.

Member Name

Specifies the name of the server in the cluster. On most platforms, the name of the server is the process name. The name must match the (object) name of the application server.

Data type	String

Weight

Controls the amount of work directed to the application server. If the weight value for the server is greater than the weight values assigned to other servers in the cluster, then the server receives a larger share of the server workload.

Data type	Integer
Range	0 to 20

Unique ID

Specifies a numerical identifier for the application server that is unique within the cluster. The ID is used for affinity.

Data type	Integer

Weight advisors

A weight advisor generates load-balancing weights, which are indicators of relative cluster member performance. You can choose to use a weight advisor to generate weights dynamically, based on performance information collected from cluster members, or you can use the static weights that are specified for individual cluster members.

A weight advisor gathers information and generates load-balancing weights for WebSphere Application Server clusters. To do this, a weight advisor repeatedly:

- Gathers metric information.
- Uses the metric values to calculate a weight for each cluster.
- Sends the weights to the deployment manager through the clusters.

The metrics are provided by the Performance Monitoring Infrastructure (PMI), which must be enabled on each cluster member for which you want to use a weight advisor, and on the node agents on which that cluster member resides.

You can choose to use a weight advisor on one or more clusters by enabling one on the weight advisor page. By default, weight advisors are disabled.

Assigning weights to cluster members

To assist a server cluster in balancing the work loads of its cluster members, you can provide information about the weights assigned to cluster members on the Weight Advisor page.

Steps for this task

1. Click **Servers > Clusters** in the console navigation tree and then **Weight Advisor** to go to the Weight Advisor page.
2. Specify the source of weights assigned to cluster members. A server cluster uses the weights to balance the work loads of its cluster members.

   Specify **Fixed Weight** to use the static weight values assigned to cluster members.

   Specify **WebSphere Dynamic Weighting** to generate weights for cluster members dynamically based on performance information collected from cluster members. This option is only valid for routers based on IBM application servers or the Edge caching proxy server. This option does not work with standard Web servers or standard plug-in file configurations.
3. Click **OK**.
4. **(Optional)** Click **Save** on the administrative console taskbar and save your administrative configuration.

Weight advisor settings

Use this page to provide information about the weights assigned to cluster members. A server cluster uses this information to balance the work loads of its cluster members.

To view this administrative console page, click **Servers > Clusters >** *cluster_name* **> Weight Advisor**.

Weight Advisor

Specifies the source of weights assigned to cluster members. A server cluster uses the weights to balance the work loads of its cluster members. Specify **Fixed Weight** to use the static weight values assigned to cluster members. Specify **WebSphere Dynamic Weighting** to generate weights for cluster members dynamically based on performance information collected from cluster members.

The **WebSphere Dynamic Weighting** option is only valid for routers based on IBM application servers or the Edge caching proxy server. The dynamic weighting option does not work with standard Web servers or standard plug-in file configurations. Also, the dynamic weighting option requires that the Performance Monitoring Service (PMI) be enabled.

Data type	String

Troubleshooting tips for dWLM Application

A weight advisor collects information that advises each cluster member as to what its weight should be dynamically (during run time). A weight can be set by a weight advisor using the console, scripting, or the ClusterMgr MBean. To enable a weight advisor using the console, you specify **WebSphere Dynamic Weighting** when defining a weight advisor.

If the weight for a cluster member is set by a weight advisor, the dWLM Application takes the weight for the cluster member and passes the weight value back to the WLMServlet Advisor loaded in the Content Based Routing (CBR) component at the HTTP front end.

For information on the WLMServlet Advisor and the CBR component, refer to the Load Balancer Administration Guide at http://www-3.ibm.com/software/webservers/appserv/doc/v50/ec/infocenter/index.html. At that URL, click **Load Balancer Administration Guide > Content Based Routing (CBR) component > Load balancing WebSphere Application Servers (WAS)**.

This article suggests how to fix problems that might arise.

Weights are not updated in the Load Balancer's WLMServlet Advisor

Problem	The virtual host is not defined in the virtual host list for cluster members.
Recommended response	The dWLM Application for the cluster member is installed to the default virtual host. Ensure that the virtual host entries for cluster members are shown in the virtual host list. If the cluster contains Web modules that have a custom virtual host list, you can specify that the dWLM Application use that virtual host list: 1. Click **Applications > Enterprise Applications >** *dWLM_Application_for_cluster_with_virtual_host_list* **> Map virtual hosts for web modules**. 2. Select the custom-defined virtual host, and click **OK**. 3. Click **Save** to apply the changes.

dWLM Application is not installed

Problem	The name of the dWLM Application for the cluster is not shown on the Enterprise Applications page, which you access by clicking **Applications > Enterprise Applications**.
Recommended response	Check the deployment manager logs for dWLM Application installation errors. The dWLM Application (wlmservlet.ear) is under the *Deployment_Manager_installation_root*\installableApps directory. If the logs do not mention dWLM Application errors, contact IBM support.

Weights have not changed for the WLMServlet Advisor in the Load Balancer

Problem	The weights are updated only when they change. Even after updating, the weights have not changed.
Recommended response	If the weights are changed through the console, refer to ""Assigning weights to cluster members"." If the weights are changed by the weight advisor, refer to ""Backup clusters"." If weights are updated through the ClusterMgr MBean, verify that these updates are reflected on the Runtime tab for the cluster member.

dWLM Application installed without request to do so

Problem	The dWLM Application might install when another application installs, even though you do not want to use the dWLM Application or dynamic workload management.
Recommended response	To stop installation of the dWLM Application, do one of the following: 1. Rename the *Deployment_Manager_installation_root* \installableApps\wlmservlet.ear file to a different name. 2. Remove the *Deployment_Manager_installation_root* \installableApps\wlmservlet.ear file. Note that, after you remove wlmservlet.ear, every time you install an application a warning message appears indicating that the dWLM Application was not installed. Disregard this message.

Backup clusters

Version 5.0.2 of WebSphere Application Server introduces "mirrored cluster support," which provides backup clusters that mirror the primary server clusters. Mirrored cluster support is for EJB requests only.

Overview and prerequisites

When all members of a cluster are no longer available to service EJB requests, clients that must interact with one of the EJB application servers in the cluster do not function. Mirrored clusters enable an EJB cluster (primary cluster) to failover to another EJB cluster (backup cluster) when none of the EJB application servers in the primary cluster are able to service a request. The backup cluster allows the client to continue functioning when all cluster members of the primary cluster are not available.

If cluster members in the primary cluster subsequently become available, the mirrored cluster support attempts to fail back the requests to the primary cluster. This fail back is automatic, so you do not have to instruct the mirrored cluster support explicitly to allow fail back.

For the backup cluster to take over servicing requests successfully, the objects and resources available in the primary cluster must also be available in the backup cluster. You must install the same applications, use the same application names, and define the same resources in the backup cluster as in the primary cluster.

For successful fail back, the primary cluster must be defined as a backup for the backup cluster. In other words, both primary and backup clusters must have a backup configured, and each cluster's backup must point to the opposite cluster.

The primary and backup clusters must have the same cluster name and the same application names. Thus, the primary cluster and the backup cluster must reside in separate cells because a cluster must have a unique name within a cell.

The mirrored cluster failover support is not cell (domain) level failover or node agent failover. For the most part, the mirrored cluster support depends on a running deployment manager and node agents, and is limited to cluster failover only. For example, if the primary cluster's entire cell stops processing, a new client's requests to the primary cluster will fail and are not sent to the backup cluster. Information regarding the backup cluster cannot be retrieved from the primary cluster's cell because the primary cluster's cell is not processing. On the other hand, if the primary cluster's cell manager or node agents and application servers stop processing after the client has already been sending requests to the primary cluster, the client already knows about the backup cluster and is able to send the requests to the backup cluster.

For cluster failover and fail back to function as expected, all of the servers (deployment manager, node agents and application servers) for both the primary cluster and the backup cluster must be at a release and level that provides mirrored cluster support; that is, Version 5.0.2 or later.

Configuration

The mirrored cluster support defines a mirrored cluster as a backup cluster which is optional and which a user might or might not configure. Each cluster can have only one backup cluster, which must be configured before it is specified as a backup cluster.

To configure a backup cluster in a cluster, you must specify a name and an endpoint. The endpoint is called the ″domain bootstrap address″ and consists of a bootstrap host and port. The bootstrap host is the host that contains the deployment manager in which the backup cluster is configured. The bootstrap port is this same deployment manager's bootstrap port.

The primary cluster and the backup cluster must reside in separate cells. To place mirrored clusters in separate cells, configure the appropriate backup cluster endpoint. The backup cluster bootstrap host and port determine which cell contains the backup cluster.

You can configure a backup cluster using the administrative console or the ExtendedCluster MBean. To determine the value for the domain bootstrap address and configure a backup cluster using the console, see ″Creating backup clusters.″

As to configuring a backup cluster using the console, you use the **Configuration** tab on the Domain Bootstrap Address page to statically define the backup cluster; the static value is consumed each time the deployment manager starts. You use the

Runtime tab on the Domain Bootstrap Address page to define the backup cluster during run time only; when the deployment manager stops, the run-time backup cluster information is discarded.

As to configuring a backup cluster using the ExtendedCluster MBean, you can change the run-time configuration only. The MBean change does not affect the static configuration. You can use the setBackup operation on the ExtendedCluster MBean to change the run-time configuration. For example, you can use the following Java code to set the primary cluster's backup cluster:

```
ac.invoke(extendedCluster, "setBackup", new Object[] {
  backupClusterName, backupBootstrapHost, backupBootstrapPort},
  new String[] {
    "java.lang.String", "java.lang.String", "java.lang.Integer"});
```

Where *ac* is the AdminClient, and *extendedCluster* is the ExtendedClusterObjectName for the primary cluster.

Fail back support

There are two scenarios that affect the cluster fail back support.

In the first scenario, requests are made by the client to the primary cluster, which eventually stops accepting requests. The requests are then routed to the backup cluster. The client initially sent requests to the primary cluster and therefore has information or ″knows about″ the primary cluster. As a result, when the primary cluster is available again, the requests fail back to the primary cluster.

In the second scenario, the client does not start sending requests until after the primary cluster is down, and the requests go directly to the backup cluster. In this case, the client has information or ″knows about″ only the backup cluster. Since the client knows only about the backup cluster, when the primary cluster becomes available, the requests from this client continue to route to the backup cluster and do not fail back to the primary cluster when it becomes available. This scenario occurs when an object is created on the backup cluster. In this case, the backup cluster becomes the primary cluster for this object.

Both of these scenarios can occur within the same client at the same time, if the client is sending requests to existing objects and creating new objects after the primary cluster stops processing.

Creating backup clusters

You configure a backup cluster to handle requests if the primary cluster fails.

To configure a backup cluster, you must specify a name and an endpoint. The endpoint is called a ″domain bootstrap address″ and consists of a bootstrap host and port. The bootstrap host is the host that contains the deployment manager in which the backup cluster is configured. The bootstrap port is this same deployment manager's bootstrap port.

The primary cluster and the backup cluster must reside in separate cells. The backup cluster bootstrap host and port determine which cell contains the backup cluster.

Steps for this task

1. Determine the bootstrap host and port of the backup cluster.

   a. Connect the administrative console to the deployment manager that contains the backup cluster.
   b. Click **System Administration > Deployment Manager > End Points > BOOTSTRAP_ADDRESS** to go to the (BOOTSTRAP_ADDRESS endpoint page). The host and port for the BOOTSTRAP_ADDRESS instance is the host and port that the backup cluster will use.
   c. Connect the administrative console to the deployment manager that contains the primary cluster.
2. Click **Servers > Clusters > Backup Cluster** to go to the Backup Cluster page.
3. Ensure that the name of the backup cluster is the same as the containing server cluster.
4. Click **Domain Bootstrap Address** and, on the Domain Bootstrap Address page, specify the backup cluster's deployment manager bootstrap host and port in the **Host** and **Port** fields. Then, click **OK**. The bootstrap host and port combined define a bootstrap address for the deployment manager.

   Note that, on the Domain Bootstrap Address page, you use the **Configuration** tab to statically define the backup cluster; the static value is consumed each time the deployment manager starts. You use the **Runtime** tab to define the backup cluster during run time only; when the deployment manager stops, the run-time backup cluster information is discarded.
5. Click **OK**.
6. Click **Save** on the administrative console taskbar and save your administrative configuration.

Backup cluster settings

Use this page to configure a backup server cluster. The backup server cluster is used if the primary server cluster fails.

Configuration of a backup cluster is only useful if the cluster contains an EJB module and a client outside of the cluster uses the EJB module.

You can view run-time information such as the status of the backup cluster, provided you saved your administrative configuration after configuring the backup cluster, restarted the server and the workload management configuration reads the backup cluster value when the server starts.

To view this administrative console page, click **Servers > Clusters >** *cluster_name* **> Backup Cluster**.

Backup Cluster Name

Specifies the name of the backup cluster. The backup cluster must have the same name as the server cluster containing the backup cluster. The backup cluster and its containing server cluster can have identical names because they must reside in different cells.

Data type	String

Domain bootstrap address settings

Use this page to specify the bootstrap address end point of the deployment manager that contains the backup cluster. End points provide host and port specifications.

When no host or port values are shown for the bootstrap address endpoint, the backup cluster is not set. No host or port values are shown if a user has never set values for the backup cluster or if the user defined a backup cluster with an actual host and port and then removed the backup cluster by removing all text from the host and port fields.

After you set host and port values on the **Configuration** tab, the bootstrap address endpoint values become active when you restart the server. Use the **Runtime** tab to update the bootstrap address endpoint values dynamically. The system uses the run-time values until you restart the server or change the values. The values on the two tabs are independent. That is, you can specify a run-time backup cluster that is different from the configuration backup cluster.

To view this administrative console page, click **Servers > Clusters >** *cluster_name* **> Backup Cluster > Domain Bootstrap Address**.

Host

Specifies the IP address, domain name server (DNS) host name with domain name suffix, or just the DNS host name, of the bootstrap host for the deployment manager of the backup cluster.

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

Port

Specifies the bootstrap port number for the deployment manager of the backup cluster. The port value is used in conjunction with the host name.

Data type	Integer

Replication

WebSphere Application Server provides a service that transfers data or events among WebSphere Application Server servers. The service is called *WebSphere Internal Replication*, or *replication* for short.

The replication service transfers both J2EE application data and any internal data used to maintain the application data among WebSphere run-time processes in a cluster of application servers.

Currently, the Web container in WebSphere Application Server leverages replication.

The replication service can replicate HttpSession data among processes and retrieve the HttpSession if the process that currently maintains the HttpSession fails. Using replication for HttpSession failover provides a potentially lower cost and more easily administrable alternative to storing HttpSession in a relational database. Further, the service can distribute across a WebSphere cluster information on invalid data and actual cached data maintained by a Web container's dynamic caching.

Replication entry

A replication entry (or *replicator*) is a run-time component that handles the transfer of internal WebSphere Application Server data.

WebSphere Application Server processes can connect to any replicator within a domain to receive data from other processes connected to any other replicator in the same domain. If the replicator a process is connected to goes down, the WebSphere Application Server process automatically attempts to reconnect to another replicator in the domain and recover data missed while unconnected.

You can define replicators to operate within a running application server process. Replicators are not enabled by default. You must define replicators as needed as part of application server and cluster management.

You can take the default settings for replicators or specify settings values for replicators that better suit your server configuration. The default configuration options are suitable for many scenarios.

Replication domain

A replication domain is a collection of replicator entry (or *replicator*) instances used by clusters or individual servers within a cell.

All replicators within a replication domain connect with each other, forming a network of replicators.

The default is to define a replication domain for a cluster when creating the cluster. However, replication domains can span across clusters.

Global default settings apply to all replication use for a given replication domain across a cell. Most default settings tune and control the behavior of replicator entries in managed servers across the cell. Such default settings control the use of encryption or the serialization and transferring of objects. Some default settings tune and control how specific WebSphere Application Server functions (for example, session manager and dynamic caching) leverage replication, such as session use of partitions.

For situations that require settings values other than the default, change the values for a given replication domain on the Internal Replication Domains page. Settings include various resource allocation, replication strategies (such as grouping or partitioning) and methods, as well as some security related items.

If you are using replication for HttpSession failover, you might need to filter where the session replicates to. For example, only replicate to two places out of many. The global default settings define the partition size or number of groups and the session manager settings define the groups to which a particular instance belongs.

Filtering is less important if you are using replication to distribute information on invalid data and actual cached data maintained by a Web container’s dynamic caching. Replication does not occur for failover as much as for data synchronization across a cluster or cell when you likely want to avoid expensive costs for generating data potentially needed across those various servers.

Note that you can filter or segment by using multiple replication domains.

Replicating data

To enable the sharing of data among processes and the backing up of failed processes, you can use the replication service provided by WebSphere Application Server. To use the service, you define replication domains, which list interconnected replicator entries (residing in managed servers in the cell) that can exchange data.

There are two ways to define replication domains and replicator entries:

- You can use the Internal Replication Domains page and Replicator Entry page to define replication domains and replicator entries. To access the Internal Replication Domains page, click **Environment > Internal Replication Domains** in the console navigation tree. To access the Replicator Entry page, click a replication domain on the Internal Replication Domains page and then click **Replicator Entries**. When you create the entries on the Replicator Entry page, you can select any server for the replicator to reside in. The page lists all servers in the cell that do not already have replicators defined.
- You can define replication domains and replicator entries when you create a cluster on the Create New Cluster page. Using the page allows you to create a replication domain that has the same name as the cluster and, as you add or create new application servers in the cluster, define replicator entries in those servers. To access the Create New Cluster page, click **Servers > Clusters** in the console navigation tree to go to the Server Clusters page and click **New**.

A replicator does not need to run in the same process as the application server that uses it. However, it might be easier to manage replicators and replication domains if a one-to-one correspondence exists between replicators and application servers. During configuration, an application server connects by default to its local replicator, so you do not need to explicitly specify the replicator to use. All processes share equally in the replication cost.

Steps for this task

1. Follow the instructions to “Creating application servers” on page 4. Later, enable a replication domain and its replicators (step 2).

   Or, create a cluster and add an application server to it. When you define the cluster, you can specify that you want a replication domain associated with the cluster. Also, when you define a cluster, you can specify that you want a replicator associated with an application server. For example, you might specify that a replicator launch in the same Java virtual machine as a Web container. Or, you can enable a replicator later (step 2).
2. Create a replication domain if one is not already created for the processes you want supported by data replication. Go to the Replication Domains page and click **New**. On the settings for a replication domain instance, specify values for the instance. The default values generally will be sufficient, especially as to pooling and timeout values.
   a. Name the replication domain.
   b. Specify the timeout interval.
   c. Specify the encryption type. The DES and TRIPLE_DES options encrypt data sent between WebSphere Application Server processes and better secure the network joining the processes.
   d. Partition the replication domain to filter the number of processes to which data is sent. Partitioning the replication domain is most often done if you are replicating data to support retrieval of an HttpSession if the process

maintaining the HttpSession fails. Partitioning is not supported for sharing of cached data maintained by Web container dynamic caching.

  e. Specify whether you want a single replication of data to be made. Enable the option if you are replicating data to support retrieval of an HttpSession if the process maintaining the HttpSession fails.
  f. Specify whether processes should receive data in objects or bytes. Processes receiving data in objects receive the data and class definitions. Processes receiving data in bytes receive the data only.
  g. **(Optional)** Configure a pool of replication resources. Pooling replication resources can enhance the performance of the internal data replication service.

3. Create replicators for the processes you want supported by data replication, if replicators have not already been created for the processes. The default convention is to define a replicator in each application server that uses replication. However, you can define a pool of replicators, separate from the servers hosting applications.
   a. Click on the replication domain instance on the Replication Domains page and then **Replicator Entries** to access the Replicator Entry page.
   b. Click **New** and, on the replicator entry settings page, define a replicator. Specify a replicator name and, from the drop-down list of the available servers within the cell to which you can assign a replicator, select a server. Also specify a host name and ports. Note that a replicator has two end points (replicator and client end points) that use the same host name but have different ports.
4. If you use the DES or TRIPLE_DES encryption type for a replicator, click **RegenerateKey** on the settings for a replication domain instance at regular intervals, such as monthly.

   Periodically changing the key enhances security.

Internal replication domain collection

Use this page to view and manage replicator instances used within a cell. Replicators can transfer both application data and any internal data used to maintain the application data among WebSphere Application Server run-time processes in a cluster of application servers.

To view this administrative console page, click **Environment > Internal Replication Domains**.

Using replicators, you can replicate HttpSession data among processes and retrieve the HttpSession if the process that currently maintains the HttpSession fails. Further, you can distribute across a cell the creation, modification, and invalidation of cached data maintained by a Web container's dynamic caching.

If you are using replication for HttpSession failover, you will likely need to filter where the session replicates to. For example, only replicate to two places out of many. The global default settings define the partition size or number of groups and the session manager settings define the groups to which a particular instance belongs. Filtering is less important if you are using replication to distribute information on cached data maintained by a Web container's dynamic caching.

The default is to define a replication domain for a cluster when creating the cluster. However, you can create a new domain from this page. Click **New** and follow the instructions on the page displayed.

Clicking **Delete** deletes a domain and all replicators defined under the domain.

Name

Specifies a name for the replication domain.

Internal replication domain settings

Use this page to configure a replicator instance.

To view this administrative console page, click **Environment > Internal Replication Domains >** *replication_domain_name*.

An application server connected to replicator within a domain can access the same set of data sent out by any application server connected to any other replicator (including the same replicator). Data is not shared across replicator domains.

Name

Specifies a name for the replication domain.

Data type	String

Request Timeout

Specifies the number of seconds that a replicator waits when requesting information from another replicator before giving up and assuming the information does not exist. The default is 5 seconds.

Data type	Integer
Units	Seconds
Default	5

Encryption Type

Specifies the type of encryption used before transfer. The options include NONE, DES, TRIPLE_DES. The default is NONE. The DES and TRIPLE_DES options encrypt data sent between WebSphere processes and better secure the network joining the processes.

If you specify DES or TRIPLE_DES, a key for global data replication is generated after you click **Apply** or **OK**. When you use the DES or TRIPLE_DES encryption type, click **RegenerateKey** at regular intervals such as monthly because periodically changing the key enhances security.

Data type	String
Default	NONE

DRS Partition Size

Specifies the number of groups into which a replication domain is partitioned. By default, data sent by a WebSphere Application Server process to a replication domain is transferred to all other WebSphere Application Server processes connected to that replication domain. To filter or reduce the number of destinations for the data being sent, partition the replication domain. The default partition size is 10, and the partition size should be 10 or more to enhance performance.

Partitioning the replication domain is most often done if you are replicating data to support retrieval of an HttpSession if the process maintaining the HttpSession fails. Partitioning is not supported for sharing of cached data maintained by Web container dynamic caching. As to dynamic caching, all partitions or groups are always active and used for data replication.

When you partition a replication domain, you define the total number of groups or partitions. Use this setting to define the number of groups. Then, when you configure a specific session manager under a Web container or as part of an enterprise application or Web module, select the partition to which that session manager instance listens and from which it accepts data. To specify the groups to which an application server listens, change the settings for affected servers on a Session Manager page. In addition, you can set a *replicator role* for a server. This replicator role affects whether a WebSphere process sends data to the replication domain, receives data, or does both. The default is both to receive and send data.

Data type	Integer
Default	10

Single Replica

Specifies that a single replication of data be made. Enable this option if you are replicating data to support retrieval of an HttpSession if the process maintaining the HttpSession fails. This option restricts the recipient of the data to a single instance.

This setting provides filtering beyond grouping or partitioning. Using this setting, you can choose to have data only sent to one other listening instance in the replication domain.

Data type	Boolean
Default	false

Serialization Method

Specifies the object serialization method to use when replicating data. An administrative concern with replicating Java objects is locating the class definition, especially in a J2EE environment where class definitions might reside only in certain web modules or enterprise applications. Object serialization methods define whether the processes receiving data also need the class definition.

The options for this setting are OBJECT and BYTES. The default is BYTES.

OBJECT instructs a replicator to write the object directly to the stream. With OBJECT, a replicator must reinstantiate the object on the receiving side so must have the class definition.

BYTES instructs a replicator to break down the object into bytes and then send only the bytes across the stream. With BYTES, a replicator does not need to instantiate the object on the receiving side. The BYTES option is useful for failover, where the data is not used at the receiving side and thus the class definitions do not need to be stored there. Or, the option requires that you move class definitions from the Web application class path to the system class path.

Data type	String
Default	BYTES

Range	Valid values are OBJECT or BYTES.

DRS Pool Size

Specifies the maximum number of items allowed in a pool of replication resources. The default is 10.

Pooling replication resources can enhance the performance of the WebSphere internal data replication service.

Data type	Integer
Default	10
Range	1 to 50

DRS Pool Connections

Specifies whether the data replication service includes replicator connections in a pool of replication resources. Whether this option is enabled or not, the pool includes replicator sessions, publishers and subscribers.

The default is not to include replicator connections in the pool.

Data type	Boolean
Default	false

Replicator entry collection

Use this page to view and manage replicator entries.

To view this administrative console page, click **Environment > Internal Replication Domains >** *replication_domain_name* **> Replicator Entries**.

To configure a new replicator entry, click **New** and follow the instructions on the page. You add a replicator to an existing server in the cell.

Replicator Name

Specifies a name for the replicator entry.

Replicator entry settings

Use this page to view and configure a replicator entry (or *replicator*).

To view this administrative console page, click **Environment > Internal Replication Domains >** *replication_domain_name* **> Replicator Entries >** *replicator_entry_name*.

Replicators communicate using TCP/IP. Thus, you must allocate an IP address and ports for replicators. Use this page to name a replicator and then to allocate an IP address and two ports (replicator and client ports) for the replicator.

Replicator Name

Specifies a name for the replicator entry.

Data type	String

Server

Specifies the server for which you are defining a replicator. The drop-down list provides the names of servers that do not already have replicators.

Data type	String
Default	None

Replicator and Client Host Name

Specifies the IP address, DNS host name with domain name suffix, or just the DNS host name, used by a client to request a Web application resource (such as a servlet, JSP file, or HTML page).

A replicator port and client port share the same host name.

Data type	String
Default	None

Replicator Port

Specifies the port for which the Web server is configured to accept client requests. The port value is used in conjunction with the host name.

The replicator port enables communication among replicators. It provides replicator port to replicator communication. The usual value specified is 7874.

Data type	Integer
Default	None

Client Port

Specifies the port for which the Web server is configured to accept client requests. The port value is used in conjunction with the host name.

The client port enables communication between an application server process and a replicator. It provides client port to application server communication. The usual value specified is 7873.

Data type	Integer
Default	None

Replication

WebSphere Application Server provides a service that transfers data or events among WebSphere Application Server servers. The service is called *WebSphere Internal Replication*, or *replication* for short.

The replication service transfers both J2EE application data and any internal data used to maintain the application data among WebSphere run-time processes in a cluster of application servers.

Currently, the Web container in WebSphere Application Server leverages replication.

The replication service can replicate HttpSession data among processes and retrieve the HttpSession if the process that currently maintains the HttpSession fails. Using replication for HttpSession failover provides a potentially lower cost and more easily administrable alternative to storing HttpSession in a relational database. Further, the service can distribute across a WebSphere cluster information on invalid data and actual cached data maintained by a Web container's dynamic caching.

Replication entry

A replication entry (or *replicator*) is a run-time component that handles the transfer of internal WebSphere Application Server data.

WebSphere Application Server processes can connect to any replicator within a domain to receive data from other processes connected to any other replicator in the same domain. If the replicator a process is connected to goes down, the WebSphere Application Server process automatically attempts to reconnect to another replicator in the domain and recover data missed while unconnected.

You can define replicators to operate within a running application server process. Replicators are not enabled by default. You must define replicators as needed as part of application server and cluster management.

You can take the default settings for replicators or specify settings values for replicators that better suit your server configuration. The default configuration options are suitable for many scenarios.

Replication domain

A replication domain is a collection of replicator entry (or *replicator*) instances used by clusters or individual servers within a cell.

All replicators within a replication domain connect with each other, forming a network of replicators.

The default is to define a replication domain for a cluster when creating the cluster. However, replication domains can span across clusters.

Global default settings apply to all replication use for a given replication domain across a cell. Most default settings tune and control the behavior of replicator entries in managed servers across the cell. Such default settings control the use of encryption or the serialization and transferring of objects. Some default settings tune and control how specific WebSphere Application Server functions (for example, session manager and dynamic caching) leverage replication, such as session use of partitions.

For situations that require settings values other than the default, change the values for a given replication domain on the Internal Replication Domains page. Settings include various resource allocation, replication strategies (such as grouping or partitioning) and methods, as well as some security related items.

If you are using replication for HttpSession failover, you might need to filter where the session replicates to. For example, only replicate to two places out of many. The global default settings define the partition size or number of groups and the session manager settings define the groups to which a particular instance belongs.

Filtering is less important if you are using replication to distribute information on invalid data and actual cached data maintained by a Web container's dynamic caching. Replication does not occur for failover as much as for data synchronization across a cluster or cell when you likely want to avoid expensive costs for generating data potentially needed across those various servers.

Note that you can filter or segment by using multiple replication domains.

Starting clusters

You can start all members of a cluster at the same time by requesting that the state of a cluster change to *running*. That is, you can start all application servers in a server cluster at the same time.

When you request that all members of a cluster start, the cluster state changes to *websphere.cluster.partial.start* and each server that is a member of that cluster launches, if it is not already running. After all members of the cluster are running, the cluster state becomes *websphere.cluster.running*.

Steps for this task

1. Click **Servers > Clusters** in the console navigation tree to access the Server Cluster page.
2. Put a checkmark in the check boxes beside those clusters whose members you want started.
3. Click **Start** or **RippleStart**.
   - **Start** launches the server process of each member of the cluster by calling the node agent for each server to start the servers. After all servers are running, the state of the cluster changes to *websphere.cluster.running*. If the call to a node agent for a server fails, the server will not start.
   - **RippleStart** combines stopping and starting operations. It first stops and then restarts each member of the cluster.

Note to Windows users: If you start and stop application servers that are part of a cluster using the Windows Services facility, the cluster state does not always update correctly. For example, if a cluster is running and you stop a cluster member through the Services GUI, the cluster state remains as *Started* even though the server is no longer running.

Stopping clusters

You can stop all members of a cluster at the same time by requesting that the state of a cluster change to *stopped*. That is, you can stop all application servers in a server cluster at the same time.

Steps for this task

1. Click **Servers > Clusters** in the console navigation tree to access the Server Cluster page.
2. Put a checkmark in the check boxes beside those clusters whose members you want stopped.
3. Click **Stop** or **Immediate Stop**.
   - **Stop** halts each server in a manner that allows the server to finish existing requests and allows failover to another member of the cluster. When the stop operation begins the cluster state changes to *websphere.cluster.partial.stop*. After all servers stop, the cluster state becomes *websphere.cluster.stopped*.

- **Immediate Stop** brings down the server quickly without regard to existing requests. When the stop operation begins, the cluster state changes to *websphere.cluster.partial.stop*. After all servers stop, the cluster state becomes *websphere.cluster.stopped*.

You can also stop and start server clusters from the settings page for a server cluster instance. To access such a page, click on the server cluster that you want to start or stop in the collection under **Name** on a Server Cluster page. You can view the status of a server cluster (that is, whether the cluster is started or stopped) on the **Runtime** tab of the settings page for a server cluster instance. Note that the **Runtime** tab is only shown if you have clicked **Save** on the administrative console taskbar since creating the server cluster instance.

Note to Windows users: If you start and stop application servers that are part of a cluster using the Windows Services facility, the cluster state does not always update correctly. For example, if a cluster is running and you stop a cluster member through the Services GUI, the cluster state remains as *Started* even though the server is no longer running.

Tuning a workload management configuration

You can set values for several workload management client properties to tune the behavior of the workload management run time. You set the properties as command-line arguments for the Java virtual machine (JVM) process in which the workload management client is running.

Caution: Set the values of these properties only in response to problems that you encounter. In most cases, you do not need to change the values. If workload management is functioning correctly, changing the values can produce undesirable results.

To change the property values, you can use the Java Virtual Machine page of the administrative console or use the wsadmin tool. In cases such as where a servlet is a client to an enterprise bean, use the administrative console page for the application server where the servlet is running to configure the properties. The steps below describe how to change the values using the console.

Steps for this task

1. Access the "Java virtual machine settings" on page 32 page.
   a. Click **Servers > Application Servers** in the console navigation tree.
   b. On "Application server collection" on page 6 click on the name of the server where the client is running.
   c. On "Application server settings" on page 6, click **Process Definition**.
   d. On "Process definition settings" on page 27, click **Java Virtual Machine**.
2. On "Java virtual machine settings" on page 32, specify one or more of the following command-line arguments in the **Generic JVM arguments** field:

   `-Dcom.ibm.CORBA.RequestTimeout=timeout_interval`
   If your application is experiencing problems with timeouts, this argument changes the value for the com.ibm.CORBA.RequestTimeout property, which specifies the timeout period for responding to requests sent from the client. This argument uses the -D option. *timeout_interval* is the timeout period in seconds. If your network experiences extreme latency, specify a large value to prevent timeouts. If you specify a value

that is too small, an application server that participates in workload management can time out before it receives a response.

Note: Be careful specifying this property; it has no recommended value. Set it only if your application is experiencing problems with timeouts.

`-Dcom.ibm.websphere.wlm.unusable.interval=interval`
If the workload management state of the client is refreshing too soon or too late, this argument changes the value for the com.ibm.websphere.wlm.unusable.interval property, which specifies the time interval that the workload management client run time waits after it marks a server as unavailable before it attempts to contact the server again. This argument uses the -D option. *interval* is the time in seconds between attempts. The default value is 300 seconds. If the property is set to a large value, the server is marked as unavailable for a long period of time. This prevents the workload management refresh protocol from refreshing the workload management state of the client until after the time period has ended.

3. Click **OK**.
4. Stop the Application Server. and then Restart the application server..

Workload management run-time exceptions

The workload management service can throw the following exceptions if it encounters problems:

org.omg.CORBA.TRANSIENT with a minor code 1229066306 (0x40421042)
This exception is thrown if the workload management routing service cannot retry a request and the failure resulted from a connection error. This exception indicates that the application should invoke some compensation logic and resubmit the request.

org.omg.CORBA.NO_IMPLEMENT with a minor code 1229066304 (0x49421040)
This exception is thrown if the workload management service cannot contact any of the EJB application servers that participate in workload management.

The WebSphere Application Server client can catch these exceptions and then implement its own strategies to handle the situation. For example, it can display an error message if no servers are available.

The workload management routing service can reroute a failed request to a different target transparently to the application if the application will not be adversely affected by a second attempt. Currently, the only way is to check if the request did not execute in whole or part on the previous attempt. When a request executes in whole or in part, an *org.omg.CORBA.TRANSIENT with the minor code 1229066306 (0x49421042)* exception is thrown to signal that a request can be made again. This informs the application that another target might be available to satisfy the request, but the request could not be failed over transparently to the application. Thus, the application can resubmit the request. The routing service throws an *org.omg.CORBA.NO_IMPLEMENT with the minor code 1229066304 (0x49421040)* exception if it cannot locate a suitable target for the request. The exception is thrown, for example, if the cluster is stopped or if the application does not have a path to any of the cluster members.

Clustering and workload management: Resources for learning

Use the following links to find relevant supplemental information about clustering and workload management. 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 model and decisions

- **Improving availability by clustering the WebSphere Application Server** (http://www-106.ibm.com/developerworks/ibm/library/i-extreme18/?open&l=937,t=gr)
- **Redbook on WebSphere Scalability: WLM and Clustering Using WebSphere Application Server Advanced Edition** (http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246153.html?Open)
- **Failover and Recovery in WebSphere Application Server Advanced Edition 4.0** (http://www7.software.ibm.com/vadd-bin/ftpdl?1/vadc/wsdd/pdf/modjeski.pdf)

Programming instructions and examples

- **WebSphere Application Server education** (http://www.ibm.com/software/webservers/learn/)
- **Listing of all IBM WebSphere Application Server Redbooks** (http://publib-b.boulder.ibm.com/Redbooks.nsf/Portals/WebSphere)
- **Redbook on IBM WebSphere V4.0 Advanced Edition Scalability and Availability** (http://publib-b.boulder.ibm.com/Redbooks.nsf/9445fa5b416f6e32852569ae006bb65f/ff31072025dcf5de85256ac