Oracle® Fusion Middleware

Administering Server Environments for Oracle WebLogic Server

14c (14.1.2.0.0)
F70333-01
December 2024

ORACLE

Oracle Fusion Middleware Administering Server Environments for Oracle WebLogic Server, 14c (14.1.2.0.0)

F70333-01

Contents

2 Configuring Web Server Functionality

3 Using Work Managers to Optimize Scheduled Work

6 Using the Batch Runtime

Preface

This document describes how you design, configure, and manage WebLogic Server environments.

Audience

This document is a resource for system administrators and operators responsible for implementing a WebLogic Server installation. It is assumed that the reader is familiar with Jakarta EE and Web technologies, object-oriented programming techniques, and the Java programming language.

This document is relevant to all phases of a software project, from development through test and production phases.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documentation

- *Understanding Oracle WebLogic Server*
- *Understanding Domain Configuration for Oracle WebLogic Server*
- *Administering Server Startup and Shutdown for Oracle WebLogic Server*
- *Oracle WebLogic Remote Console Online Help*

New and Changed WebLogic Server Features

For a comprehensive listing of the new WebLogic Server features introduced in this release, see What's New in Oracle WebLogic Server.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Configuring Network Resources

Oracle WebLogic Server configurable network resources such as network channels and domain-wide administration ports help you effectively use the network features of the machines that host your applications and manage quality of service.

- Overview of Network Configuration
  For many development environments, configuring WebLogic Server network resources is simply a matter of identifying a Managed Server listen address and listen port. However, in most production environments, administrators must balance finite network resources against the demands placed upon the network. The task of keeping applications available and responsive can be complicated by specific application requirements, security considerations, and maintenance tasks, both planned and unplanned.
- Understanding Network Channels
  Learn about network channels, the standard channels that WebLogic Server pre-configures, and common applications for channels.
- Configuring a Channel
  Use the WebLogic Remote Console or `NetworkAccessPointMBean` to configure a network channel.
- Assigning a Custom Channel to an EJB
  After you configure a custom channel, assign it to an EJB using the `network-access-point` element in `weblogic-ejb-jar.xml`, you can assign a custom channel to an EJB.
- Using IPv6 with IPv4
  WebLogic Server supports host machines that are configured to use either Internet Protocol (IP) versions 4 or 6 (IPv4 and IPv6).

Overview of Network Configuration

For many development environments, configuring WebLogic Server network resources is simply a matter of identifying a Managed Server listen address and listen port. However, in most production environments, administrators must balance finite network resources against the demands placed upon the network. The task of keeping applications available and responsive can be complicated by specific application requirements, security considerations, and maintenance tasks, both planned and unplanned.

WebLogic Server lets you control the network traffic associated with your applications in a variety of ways, and configure your environment to meet the varied requirements of your applications and end users. You can:

- Designate the Network Interface Cards (NICs) and ports used by Managed Servers for different types of network traffic.
- Support multiple protocols and security requirements.
- Specify connection and message time-out periods.
- Impose message size limits.

You specify these and other connection characteristics by defining a network channel—the primary configurable WebLogic Server resource for managing network connections. You

configure a network channel in the WebLogic Remote Console (see Configure Network Connections) or by using the `NetworkAccessPointMBean`.

Understanding Network Channels

Learn about network channels, the standard channels that WebLogic Server pre-configures, and common applications for channels.

What Is a Channel?

A network channel is a configurable resource that defines the attributes of a network connection to WebLogic Server. For instance, a network channel can define:

- The protocol the connection supports.
- The listen address.
- The listen ports for secure and non-secure communication.
- Connection properties such as the login time-out value and maximum message sizes.
- Whether or not the connection supports tunneling.
- Whether the connection can be used to communicate with other WebLogic Server instances in the domain, or used only for communication with clients.

Rules for Configuring Channels

Follow these guidelines when configuring a channel.

- You can assign a particular channel to only one server instance.
- You can assign multiple channels to a server instance.
- Each channel assigned to a particular server instance must have a unique combination of listen address, listen port, and protocol.
- You can configure a custom identity keystore, and other channel-specific SSL attributes, that are separate from and that override the default keystore and SSL configuration settings for the Managed Server instance or the domain.
- If you assign non-SSL and SSL channels to the same server instance, make sure that they do not use the same port number.

Custom Channels Can Inherit Default Channel Attributes

If you do not assign a channel to a server instance, it uses the WebLogic Server default channel, which is automatically configured by WebLogic Server, based on the attributes in

`ServerMBean` or `SSLMBean`; the operating system determines the network interface. The default channel is described in The Default Network Channel.

`ServerMBean` and `SSLMBean` represent a server instance and its SSL configuration. When you configure a server instance listen address, listen port, and SSL listen port, using the **Environment**: **Servers**: *myServer*: **General** page in the WebLogic Remote Console, those values are stored in the `ServerMBean` and `SSLMBean` for the server instance.

If you do not specify a particular connection attribute in a custom channel definition, the channel inherits the value specified for the attribute in `ServerMBean`. For example, if you create a channel, and do not define its listen address, the channel uses the listen address defined in `ServerMBean`. Similarly, if a Managed Server cannot bind to the listen address or listen port configured in a channel, the Managed Server uses the defaults from `ServerMBean` or `SSLMBean`.

Why Use Network Channels?

You use network channels to manage quality of service, meet varying connection requirements, and improve utilization of your systems and network resources. For example, network channels allow you to:

- **Segregate different types of network traffic**—You can configure whether or not a channel supports outgoing connections. By assigning two channels to a server instance—one that supports outgoing connections and one that does not—you can independently configure network traffic for client connections and server connections, and physically separate client and server network traffic on different listen addresses or listen ports.

  You cannot create an outbound only network channel; there always has to be a corresponding inbound interface, port, and protocol associated with the channel. However, you can avoid directing your traffic to it or use a firewall to block it. Also remember that a custom channel is protocol specific, so you will need a network channel defined per protocol (HTTP, HTTPS, t3, t3s, and such). See, also `NetworkAccessPointMBean.OutboundEnabled`.

  You can also segregate instance administration and application traffic by configuring a domain-wide administration port or administration channel. See Administration Port and Administrative Channel.

- **Support varied application or user requirements on the same Managed Server—**You can configure multiple channels on a Managed Server to support different protocols, or to tailor properties for secure versus non-secure traffic.

  By configuring a network channel to use a custom identity keystore, you can assert an identity on that channel that is different from the identity configured for the Managed Server or domain.

- **Segregate internal application network traffic**—You can assign a specific channel to a an EJB.

If you use a network channel with a server instance on a multihomed machine, you must enter a valid listen address either in `ServerMBean` or in the channel. If the channel and `ServerMBean` listen address are blank or specify the localhost address (IP address 0.0.0.0 or 127.*.*.*), the server binds the network channel listen port and SSL listen ports to all available IP addresses on the multihomed machine. See The Default Network Channel for information on setting the listen address in `ServerMBean`.

- Handling Channel Failures
- Upgrading Quality of Service Levels for RMI

Handling Channel Failures

When initiating a connection to a remote server, and multiple channels with the same required destination, protocol and quality of service exist, WebLogic Server will try each in turn until it successfully establishes a connection or runs out of channels to try.

Upgrading Quality of Service Levels for RMI

For RMI lookups only, WebLogic Server may upgrade the service level of an outgoing connection. For example, if a T3 connection is required to perform an RMI lookup, but an existing channel supports only T3S, the lookup is performed using the T3S channel.

This upgrade behavior does not apply to server requests that use URLs, since URLs embed the protocol itself. For example, the server cannot send a URL request beginning with `http://` over a channel that supports only `https://`.

Standard WebLogic Server Channels

WebLogic Server provides pre-configured channels that you do not have to explicitly define.

- Default channel—Every Managed Server has a default channel.
- Administrative channel—If you configure a domain-wide administration port, WebLogic Server configures an administrative channel for each Managed Server in the domain.
- The Default Network Channel
- Administration Port and Administrative Channel

The Default Network Channel

Every WebLogic Server domain has a default channel that is generated automatically by WebLogic Server. The default channel is based on the listen address and listen port defined in the `ServerMBean` and `SSLMBean`. It provides a single listen address, one port for HTTP (non-secure) communication (7001 by default), and one port for HTTPS (secure) communication (7002 by default). In the Remote Console, you can configure the listen address and listen port on the **Environment**: **Servers**: *myServer*: **General** page; the values you assign are stored in attributes of the `ServerMBean` and `SSLMBean`.

The default configuration may meet your needs if:

- You are installing in a test environment that has simple network requirements.
- Your server uses a single NIC, and the default port numbers provide enough flexibility for segmenting network traffic in your domain.

Using the default configuration ensures that third-party administration tools remain compatible with the new installation, because network configuration attributes remain stored in `ServerMBean` and `SSLMBean`.

Even if you define and use custom network channels for your domain, the default channel settings remain stored in `ServerMBean` and `SSLMBean`, and are used if necessary to provide connections to a server instance.

> **Note:**
>
> Unless specified, WebLogic Server uses the non-secure default channel for cluster communication to send session information among cluster members. If you disable the non-secure channel, there is no other channel available *by default* for the non-secure communication of cluster session information. To address this, you can:
>
> - Enable the `secureReplicationEnabled` attribute of the `ClusterMBean` so that the cluster uses a secure channel for communication. See Configuring a Replication Channel.
> - Create a custom channel for non-secure communication. See Custom Channels Can Inherit Default Channel Attributes.

Administration Port and Administrative Channel

You can separate administration traffic from application traffic in your domain by defining an optional administration port. When configured, the administration port is used by each Managed Server in the domain exclusively for communication with the domain Administration Server. If an administration port is enabled, WebLogic Server automatically generates an administrative channel for your domain, based on the port settings upon server instance startup. The administrative channel provides a listen address and listen port to handle administration traffic.

- Administration Port Capabilities
- Administration Port Restrictions
- Administration Port Requires SSL
- Configure Administration Port
- Booting Managed Servers to Use Administration Port
- Booting Managed Servers to Use Administrative Channels
- Custom Administrative Channels

Administration Port Capabilities

An administration port enables you to:

- Start a server in standby state. This allows you to administer a Managed Server, while its other network connections are unavailable to accept client connections. See STANDBY State in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.
- Separate administration traffic from application traffic in your domain. In production environments, separating traffic ensures that critical administration operations (starting and stopping servers, changing a server's configuration, and deploying applications) do not compete with high-volume application traffic on the same network connection.
- Administer a deadlocked server instance using WLST. If you do not configure an administration port, administrative commands such as `threadDump` and `shutdown` will not work on deadlocked server instances.

Administration Port Restrictions

The administration port accepts only secure, SSL traffic, and all connections via the port require authentication. Enabling the administration port imposes the following restrictions on your domain:

- The Administration Server and all Managed Servers in your domain must be configured with support for the SSL protocol. Managed Servers that do not support SSL cannot connect with the Administration Server during startup—you will have to disable the administration port in order to configure them.
- Because all server instances in the domain must enable or disable the administration port at the same time, you configure the administration port at the domain level. You can change an individual Managed Server administration port number, but you cannot enable or disable the administration port for an individual Managed Server. The ability to change the port number is useful if you have multiple server instances with the same listen address.
- After you enable the administration port, you must establish an SSL connection to the Administration Server in order to start any Managed Server in the domain. This applies whether you start Managed Servers manually, at the command line, or using Node Manager. For instructions to establish the SSL connection, see Administration Port Requires SSL.
- After enabling the administration port, all WebLogic Remote Console traffic *must* connect via the administration port.
- If multiple server instances run on the same computer in a domain that uses a domain-wide administration port, you must either:
  - Host the server instances on a multihomed machine and assign each server instance a unique listen address, or
  - Override the domain-wide port on all but one of the servers instances on the machine. Override the port using the **Local Administration Port Override** option on the **Environment**: **Servers**: *myServer*: **General** page in the WebLogic Remote Console.

Administration Port Requires SSL

The administration port requires SSL, which is enabled by default when you install WebLogic Server. If SSL has been disabled for any server instance in your domain, including the Administration Server and all Managed Servers, re-enable it using the **Environment**: **Servers**: *myServer*: **General** page in the WebLogic Remote Console.

Ensure that each server instance in the domain has a configured default listen port or default SSL listen port. The default ports are those you assign on the **Environment**: **Servers**: *myServer*: **General** page in the WebLogic Remote Console. A default port is required in the event that the server cannot bind to its configured administration port. If an additional default port is available, the server will continue to boot and you can change the administration port to an acceptable value.

By default WebLogic Server is configured to use demonstration certificate files. To configure production security components, follow the steps in Configuring SSL in *Administering Security for Oracle WebLogic Server*.

Configure Administration Port

Enable the administration port as described in Configure the Domain-Wide Administration Port in the *Oracle WebLogic Remote Console Online Help*.

After configuring the administration port, you must restart the Administration Server and all Managed Servers to use the new administration port.

Booting Managed Servers to Use Administration Port

If you reboot Managed Servers at the command line or using a start script, specify the administration port in the port portion of the URL. The URL must specify the `https://` prefix, rather than `http://`, as shown below.

```
-Dweblogic.management.server=https://host:admin_port
```

> **Note:**
>
> If you use Node Manager for restarting the Managed Servers, it is not necessary to modify startup settings or arguments for the Managed Servers. Node Manager automatically obtains and uses the correct URL to start a Managed Server.

If the hostname in the URL is not identical to the hostname in the Administration Server's certificate, disable hostname verification in the command line or start script, as shown below:

```
-Dweblogic.security.SSL.ignoreHostnameVerification=true
```

Booting Managed Servers to Use Administrative Channels

To allow a Managed Server to bind to an administrative channel during reboot, use the following command-line option:

```
-Dweblogic.admin.ListenAddress=<addr>
```

This allows the Managed Server to startup using an administrative channel. After the initial bootstrap connection, a standard administrative channel is used.

> **Note:**
>
> This option is useful to ensure that the appropriate NIC semantics are used before `config.xml` is downloaded.

Custom Administrative Channels

If the standard WebLogic Server administrative channel does not satisfy your requirements, you can configure a custom channel for administrative traffic. For example, a custom administrative channel allows you to segregate administrative traffic on a separate NIC.

To configure a custom channel for administrative traffic, configure the channel as described in Configuring a Channel, and select "admin" as the channel protocol. Note the configuration and usage guidelines described in:

- Administration Port Requires SSL
- Booting Managed Servers to Use Administration Port

Using Internal Channels

Previous version of WebLogic Server allowed you to configure multiple channels for external traffic, but required you to use the default channel for internal traffic between server instances. WebLogic Server now allows you to create network channels to handle administration traffic or communications between clusters. This can be useful in the following situations:

- Internal administration traffic needs to occur over a secure connection, separate from other network traffic.
- Other types of network traffic, for example replication data, need to occur over a separate network connection.
- Certain types of network traffic need to be monitored.

- Channel Selection
- Internal Channels Within a Cluster

Channel Selection

All internal traffic is handled via a network channel. If you have not created a custom network channel to handle administrative or clustered traffic, WebLogic Server automatically selects a default channel based on the protocol required for the connection. See The Default Network Channel.

Internal Channels Within a Cluster

Within a cluster, internal channels can be created to handle to the following types of server-to-server connections:

- Multicast traffic
- Replication traffic
- Administration traffic

See Configuring Network Channels For a Cluster.

Configuring a Channel

Use the WebLogic Remote Console or `NetworkAccessPointMBean` to configure a network channel.

For the Remote Console, see Configure Custom Network Channels. To configure a channel for clustered Managed Servers see, Configuring Network Channels For a Cluster.

For a summary of key facts about network channels, and guidelines related to their configuration, see Guidelines for Configuring Channels .

- Guidelines for Configuring Channels
- Configuring Network Channels For a Cluster

Guidelines for Configuring Channels

Follow these guidelines when configuring a channel.

- Channels and Server Instances

Channels and Server Instances

- Each channel you configure for a particular server instance must have a unique combination of listen address, listen port, and protocol.
- A channel can be assigned to a single server instance.
- You can assign multiple channels to a server instance.
- If you assign non-SSL and SSL channels to the same server instance, make sure that they do not use the same combination of address and port number.

Dynamic Channel Configuration

- In WebLogic Server, you can configure a network channel without restarting the server. Additionally, you can start and stop dynamically configured channels while the server is running. However, when you shutdown a channel while the server is running, the server does not attempt to gracefully terminate any work in progress.

Channels and Identity

- By default, when you configure a network channel, the channel uses the SSL configuration that is set for the server instance. This means that the channel uses the same identity and trust that is established for the server. The server might use a custom identity that is specific to that server, or it might be a single domain-wide identity, depending on how the server instance and domain are configured.
- You can configure a network channel to use a custom identity keystore, and other SSL attributes, that are specific to that channel. This allows you to use an identity on that channel that is different from the one configured for the server. Using this capability, you can configure a server that can switch to a different identity when communicating with a particular client.

  See Configuring an Identity Keystore Specific to a Network Channel in *Administering Security for Oracle WebLogic Server*.

Channels and Protocols

- Some protocols do not support particular features of channels. In particular the COM protocol does not support SSL or tunneling.
- You must define a separate channel for each protocol you wish the server instance to support, with the exception of HTTP.

  HTTP is enabled by default when you create a channel, because RMI protocols typically require HTTP support for downloading stubs and classes. You can disable HTTP support on the **Environment**: **Servers**: *myServer*: **Protocols**: **HTTP** page in the WebLogic Remote Console.

Reserved Names

- WebLogic Server uses the internal channel names `.WLDefaultChannel` and `.WLDefaultAdminChannel` and reserves the `.WL` prefix for channel names. Do not begin the name of a custom channel with the string `.WL`.

Channels, Proxy Servers, and Firewalls

If your configuration includes a a firewall between a proxy Web server and a cluster (as described in Firewall Between Proxy Layer and Cluster in *Administering Clusters for Oracle WebLogic Server*), and the clustered servers are configured with two custom channels for segregating HTTPS and HTTP traffic, those channels must share the same listen address. Furthermore, if both HTTP and HTTPS traffic needs to be supported, there must be a custom channel for each—it is not possible to use the default configuration for one or the other.

If either of those channels has a `PublicAddress` defined, as is likely given the existence of the firewall, both channels must define `PublicAddress`, and they both must define the same `PublicAddress`.

Configuring Network Channels For a Cluster

To configure a channel for clustered Managed Servers, note the information in Guidelines for Configuring Channels , and follow the guidelines described in the following sections.

- Create the Cluster
- Create and Assign the Network Channel
- Configuring a Replication Channel
- Increase Packet Size When Using Many Channels

Create the Cluster

If you have not already configured a cluster you can:

- Use the Configuration Wizard to create a new, clustered domain, following the instructions in Create a Clustered Domain in *Administering Clusters for Oracle WebLogic Server*, or
- Use the WebLogic Remote Console to create a cluster in an existing domain, following the instructions in Create a Cluster in the *Oracle WebLogic Remote Console Online Help*.

For information and guidelines about configuring a WebLogic Server cluster, see Before You Start in *Administering Clusters for Oracle WebLogic Server*.

Create and Assign the Network Channel

Use the instructions in Configure Network Connections in the *Oracle WebLogic Remote Console Online Help* to create a new network channel for each Managed Server in the cluster. When creating the new channels:

- For each channel you want to use in the cluster, configure the channel with the same name and protocol on each Managed Server in the cluster.

> **Note:**
>
> Failure to configure channel names and protocols identically on each server in a cluster can result in severe performance degradation when accessing RMI based objects. Examples of RMI based objects include JMS Connection Factories and EJB Homes.
>
> If an RMI object is obtained via a channel, and it attempts to connect to a server in a cluster that does not have a channel with the same name as the original channel, the object may then try to use the server's default channel. This might in turn result in unacceptable performance.
>
> While rare in practice, if the intent is to create similar but differently named channels within a cluster, then setting the server side property `weblogic.rmi.t3.replicaList.customChannel.excludeDefaultChannels=true` will result in RMI objects only attempting to accessing those channels with the exact specified name.

Note that failure to name channels identically can result in severe performance degradation when accessing JMS or EJB remote objects.

- Make sure that the listen port and SSL listen port you define for each Managed Server's channel are different than the Managed Server's default listen ports. If the custom channel specifies the same port as a Managed Server's default port, the custom channel and the Managed Server's default channel will each try to bind to the same port, and you will be unable to start the Managed Server.
- If a cluster address has been explicitly configured for the cluster, it will be appear in the Cluster Address field on the **Environment**: **Clusters**: *myCluster*: **General** page.

  If you are using dynamic cluster addressing, the Cluster Address field will be empty, and you do not need to supply a cluster address. For information about the cluster address, and how WebLogic Server can dynamically generate the cluster address, see Cluster Address in *Administering Clusters for Oracle WebLogic Server*.

> **Note:**
>
> If you want to use dynamic cluster addressing, do not supply a cluster address on the **Environment**: **Clusters**: *myCluster*: **General** page. If you supply a cluster address explicitly, that value will take precedence and WebLogic Server will not generate the cluster address dynamically.

Configuring a Replication Channel

A replication channel is a network channel that is designated to transfer replication information between clusters. If a replication channel is not explicitly defined, WebLogic Server uses a default network channel to communicate replication information.

When WebLogic Server uses a default network channel as the replication channel, it does not use SSL encryption by default. You must enable SSL encryption using the `secureReplicationEnabled` attribute of the `ClusterMBean`. You can also update this setting from the WebLogic Remote Console.

Enabling SSL encryption can have a direct impact on clustered application throughput as session replication is blocking by design. in other words, no response is sent to the client until replication is completed. You should consider this when deciding to enable SSL on the replication channel.

If a replication channel is explicitly defined, the channel's protocol is used to transmit replication traffic.

Increase Packet Size When Using Many Channels

Use of more than about twenty channels in a cluster can result in the formation of multicast header transmissions that exceed the default maximum packet size. The `MTUSize` attribute in the `Server` element of `config.xml` sets the maximum size for packets sent using the associated network card to 1500. Sending packets that exceed the value of `MTUSize` can result in a `java.lang.NegativeArraySizeException`. You can avoid exceptions that result from packet sizes in excess of `MTUSize` by increasing the value of `MTUSize` from its default value of 1500.

Assigning a Custom Channel to an EJB

After you configure a custom channel, assign it to an EJB using the `network-access-point` element in `weblogic-ejb-jar.xml`, you can assign a custom channel to an EJB.

See network-access-point in *Developing Enterprise JavaBeans, Version 3.2, for Oracle WebLogic Server*.

Using IPv6 with IPv4

WebLogic Server supports host machines that are configured to use either Internet Protocol (IP) versions 4 or 6 (IPv4 and IPv6).

If you have a domain that includes some machines that use IPv4 in network communications and others that use IPv6, and the Administration Server is hosted on a machine using IPv4, the status of the Managed Server instances hosted on the machines using IPv6 might be displayed as "unknown" in the WebLogic Remote Console.

To make the status of these Managed Server instances available in the WebLogic Remote Console, you must specify a listen address for them. If your server is running, you will have to restart it after specifying the listen address. For information on assigning the listen address for a Managed Server in an existing domain using the WebLogic Remote Console, see Specify Listen Addresses in the *Oracle WebLogic Remote Console Online Help*.

You can also specify the listen address for your Managed Server when configuring it with the Configuration Wizard. In the Remote Console, on the *myManagedServer*: **General** page, enter the physical IP address of each Managed Server in the **Listen Address** field, save changes and continue configuring.

2

Configuring Web Server Functionality

Learn how to configure a Jakarta EE Web application hosted on Oracle WebLogic Server to function as a standard HTTP Web server hosting static content. Web applications also can host dynamic content such as JSPs and servlets.

See *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

- Overview of Configuring Web Server Components
  In addition to hosting dynamic Java-based distributed applications, WebLogic Server functions as a Web server that handles high-volume Web sites, serving static files such as HTML files and image files, as well as servlets and JavaServer Pages (JSP).
- Configuring the Server
  You can specify the port that each WebLogic Server listens on for HTTP requests. Although you can specify any valid port number, if you specify port `80`, you can omit the port number from the HTTP request used to access resources over HTTP. For example, if you define port `80` as the listen port, you can use the form `http://hostname/myfile.html` instead of `http://hostname:portnumber/myfile.html`.
- Web Applications
  HTTP and Web applications are deployed according to the Jakarta Servlet 4.0 and JSP 2.3 specifications, which describe Web Applications as a standard for grouping the components of a Web-based application. These components include JSP pages, HTTP servlets, and static resources such as HTML pages or image files. In addition, a Web application can access external resources such as EJBs and JSP tag libraries. Each server can host any number of Web applications. You typically use the name of the Web application as part of the URI you use to request resources from the Web application.
- Configuring Virtual Hosting
  Virtual hosting allows you to define host names that servers or clusters respond to. When you use virtual hosting, you use DNS to specify one or more host names that map to the IP address of a WebLogic Server instance or cluster, and you specify which Web applications are served by the virtual host. When used in a cluster, load balancing allows the most efficient use of your hardware, even if one of the DNS host names processes more requests than the others.
- How WebLogic Server Resolves HTTP Requests
  When WebLogic Server receives an HTTP request, it resolves the request by parsing the various parts of the URL and using that information to determine which Web application and/or server should handle the request.
- Health Score-Based Intelligent Routing
  Health score-based intelligent routing lets you manage system optimization, avoid overload, and deal with unhealthy servers.
- Setting Up HTTP Access Logs
  WebLogic Server can keep a log of all HTTP transactions in a text file, in either *common log format* or *extended log format*.
- Preventing POST Denial-of-Service Attacks
  A Denial-of-Service attack is a malicious attempt to overload a server with phony requests. One common type of attack is to send huge amounts of data in an HTTP POST method. You can set three attributes in WebLogic Server that help prevent this type of attack. These

attributes are set in the Remote Console, under **Servers**: **Protocols**: **HTTP** or **Virtual Hosts**: **HTTP**. If you define these attributes for a virtual host, the values set for the virtual host override those set under Servers.

- Setting Up WebLogic Server for HTTP Tunneling
- Using Native I/O for Serving Static Files (Windows Only)
  When running WebLogic Server on Windows NT/2000/XP you can specify that WebLogic Server use the native operating system call `TransmitFile` instead of using Java methods to serve static files such as HTML files, text files, and image files. Using native I/O can provide performance improvements when serving larger static files.

Overview of Configuring Web Server Components

In addition to hosting dynamic Java-based distributed applications, WebLogic Server functions as a Web server that handles high-volume Web sites, serving static files such as HTML files and image files, as well as servlets and JavaServer Pages (JSP).

WebLogic Server supports the HTTP 1.1 standard.

Configuring the Server

You can specify the port that each WebLogic Server listens on for HTTP requests. Although you can specify any valid port number, if you specify port `80`, you can omit the port number from the HTTP request used to access resources over HTTP. For example, if you define port `80` as the listen port, you can use the form `http://hostname/myfile.html` instead of `http://hostname:portnumber/myfile.html`.

On UNIX systems, binding a process to a port lower than 1025 must be done from the account of a privileged user, usually root. Consequently, if you want WebLogic Server to listen on port 80, you must start WebLogic Server as a privileged user; yet it is undesirable from a security standpoint to allow long-running processes like WebLogic Server to run with more privileges than necessary. WebLogic Server needs root privileges only until the port is bound.

WebLogic Server provides capabilities to switch its UNIX user ID (UID) and/or UNIX group ID (GID) after it binds to port 80. You can change the UID (or GID) either through the WebLogic Remote Console (see Configuring the Listen Port) or by accessing `UnixMachineMBean` using WLST. Use `UnixMachineMBbean.setPostBindUID()` to set the UID and enable the switch by setting `UnixMachineMBean.setPostBindUIDEnabled()` to `true`. Similarly, the GID can be changed using methods `UnixMachineMBean.setPostBindGID()` and `UnixMachineMBean.setPostBindGIDEnabled()`.

You can switch to the UNIX account "nobody," which is the least privileged user on most UNIX systems. If desired, you may create a UNIX user account expressly for running WebLogic Server. Make sure that files needed by WebLogic Server, such as log files and the WebLogic classes, are accessible by the non-privileged user. Once ownership of the WebLogic process has switched to the non-privileged user, WebLogic will have the same read, write, and execute permissions as the non-privileged user.

You define a separate listen port for non-SSL and secure (using SSL) requests. For additional information on configuring listen ports, see Understanding Network Channels.

- Configuring the Listen Port

Configuring the Listen Port

1. Use the WebLogic Remote Console to set the listen port to port 80. See Specify Listen Ports.
2. If the machine hosting WebLogic Server is running Windows, skip to step 8.
3. Use the WebLogic Remote Console to create a new Unix Machine. See Configure Machines.
4. Select the **Enable Post-Bind UID** field.
5. Enter the user name you want WebLogic Server to run as in the **Enable Post-Bind UID** field.
6. Select the **Enable Post-Bind GID** fields.
7. Enter the group name you want WebLogic Server to run as in the **Enable Post-Bind GID** field.
8. Click **Save**.

Web Applications

HTTP and Web applications are deployed according to the Jakarta Servlet 4.0 and JSP 2.3 specifications, which describe Web Applications as a standard for grouping the components of a Web-based application. These components include JSP pages, HTTP servlets, and static resources such as HTML pages or image files. In addition, a Web application can access external resources such as EJBs and JSP tag libraries. Each server can host any number of Web applications. You typically use the name of the Web application as part of the URI you use to request resources from the Web application.

See *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

- Web Applications and Clustering

Web Applications and Clustering

Web applications can be deployed to a WebLogic Server cluster. When a user requests a resource from a Web application, the request is routed to one of the servers in the cluster that host the Web application. If an application uses a session object, then sessions must be replicated across the nodes of the cluster. Several methods of replicating sessions are provided.

See *Administering Clusters for Oracle WebLogic Server*.

Configuring Virtual Hosting

Virtual hosting allows you to define host names that servers or clusters respond to. When you use virtual hosting, you use DNS to specify one or more host names that map to the IP address of a WebLogic Server instance or cluster, and you specify which Web applications are served by the virtual host. When used in a cluster, load balancing allows the most efficient use of your hardware, even if one of the DNS host names processes more requests than the others.

For example, you can specify that a Web application called `books` responds to requests for the virtual host name `www.books.com`, and that these requests are targeted to WebLogic Servers

A,B, and C, while a Web application called `cars` responds to the virtual host name `www.autos.com` and these requests are targeted to WebLogic Servers D and E. You can configure a variety of combinations of virtual host, WebLogic Server instances, clusters, and Web applications, depending on your application and Web server requirements.

For each virtual host that you define you can also separately define HTTP parameters and HTTP access logs. The HTTP parameters and access logs set for a *virtual host* override those set for a *server*. You may specify any number of virtual hosts.

You activate virtual hosting by targeting the virtual host to a server or cluster of servers. Virtual hosting targeted to a cluster will be applied to all servers in the cluster.

- Virtual Hosting and the Default Web Application
- Setting Up a Virtual Host

Virtual Hosting and the Default Web Application

You can also designate a *default Web Application* for each virtual host. The default Web application for a virtual host responds to all requests that cannot be resolved to other Web applications deployed on the same server or cluster as the virtual host.

Unlike other Web applications, a default Web application does not use the Web application name (also called the *context path*) as part of the URI used to access resources in the default Web application.

For example, if you defined virtual host name `www.mystore.com` and targeted it to a server on which you deployed a Web application called `shopping`, you would access a JSP called `cart.jsp` from the `shopping` Web application with the following URI:

```
http://www.mystore.com/shopping/cart.jsp
```

If, however, you declared `shopping` as the default Web application for the virtual host `www.mystore.com`, you would access `cart.jsp` with the following URI:

```
http://www.mystore.com/cart.jsp
```

See How WebLogic Server Resolves HTTP Requests.

When using multiple Virtual Hosts with different default Web applications, you can not use single sign-on, as each Web application will overwrite the JSESSIONID cookies set by the previous Web application. This will occur even if the CookieName, CookiePath, and CookieDomain are identical in each of the default Web applications.

Setting Up a Virtual Host

1. Use the Remote Console to define a virtual host in the **Edit Tree**, under **Environment**: **Virtual Hosts**.
   a. **Create** a new virtual host.
   b. Configure **General** virtual host properties.
   c. Configure HTTP **Logging** settings for a virtual host.
   d. Configure **HTTP** for a virtual host.
   e. **Target** the virtual host to servers.
2. Add a line naming the virtual host to the `etc/hosts` file on your server to ensure that the virtual host name can be resolved.

How WebLogic Server Resolves HTTP Requests

When WebLogic Server receives an HTTP request, it resolves the request by parsing the various parts of the URL and using that information to determine which Web application and/or server should handle the request.

Table 2-1 demonstrates various combinations of requests for Web applications, virtual hosts, servlets, JSPs, and static files and the resulting response.

> **Note:**
>
> If you package your Web application as part of an Enterprise application, you can provide an alternate name for a Web application that is used to resolve requests to the Web application. See *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

Table 2-1 provides some sample URLs and the file that is served by WebLogic Server. The Index Directories Checked column refers to the Index Directories attribute that controls whether or not a directory listing is served if no file is specifically requested.

Table 2-1 Examples of How WebLogic Server Resolves URLs

URL	Index Directories Checked?	This file is served in response
`http://host:port/apples`	No	Welcome file* defined in the `apples` Web application.
`http://host:port/apples`	Yes	Directory listing of the top-level directory of the `apples` Web application.
`http://host:port/oranges/naval`	Does not matter	Servlet mapped with `<url-pattern>` of `/naval` in the `oranges` Web application. There are additional considerations for servlet mappings. See Configuring Servlets in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.
`http://host:port/naval`	Does not matter	Servlet mapped with `<url-pattern>` of `/naval` in the `oranges` Web application and `oranges` is defined as the default Web application. See Configuring Servlets in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.
`http://host:port/apples/pie.jsp`	Does not matter	`pie.jsp`, from the top-level directory of the `apples` Web application.

Table 2-1 (Cont.) Examples of How WebLogic Server Resolves URLs

URL	Index Directories Checked?	This file is served in response
`http://host:port`	Yes	Directory listing of the top-level directory of the *default* Web application
`http://host:port`	No	Welcome file* from the *default* Web application.
`http://host:port/apples/myfile.html`	Does not matter	`myfile.html`, from the top-level directory of the `apples` Web application.
`http://host:port/myfile.html`	Does not matter	`myfile.html`, from the top-level directory of the *default* Web application.
`http://host:port/apples/images/red.gif`	Does not matter	`red.gif`, from the images subdirectory of the top-level directory of the `apples` Web application.
`http://host:port/myFile.html` Where `myfile.html` does not exist in the `apples` Web application and a *default servlet* has not been defined.	Does not matter	Error 404
`http://www.fruit.com/`	No	Welcome file from the default Web application for a virtual host with a host name of `www.fruit.com`.
`http://www.fruit.com/`	Yes	Directory listing of the top-level directory of the `default` Web application for a virtual host with a host name of `www.fruit.com`.
`http://www.fruit.com/oranges/myfile.html`	Does not matter	`myfile.html`, from the `oranges` Web application that is targeted to a virtual host with host name `www.fruit.com`.

Health Score-Based Intelligent Routing

Health score-based intelligent routing lets you manage system optimization, avoid overload, and deal with unhealthy servers.

WebLogic Server supports intelligent load balancing, which lets Oracle HTTP Server (OHS) distribute traffic more evenly across a pool of servers according to their actual capacity. For each Managed Server in a cluster, WebLogic Server provides a default health score calculation. This health score is calculated individually, by each Managed Server instance, based on metrics and MBeans that are available locally, and then returned to OHS when requested. OHS then routes requests to the healthiest server instance. For detailed OHS information, see Support for Intelligent Load Balancing in *Using Oracle WebLogic Server Proxy Plug-Ins*.

How It Works

Each WebLogic Server instance computes its own health using a service provider plug-in. The WebLogic Health Service periodically invokes the plug-in to obtain the most current health value at a defined query interval, for example, every five seconds. OHS sends a request header, `X-WebLogic-Request-Server-Health-Score`, to each individual Managed Server instance requesting its health score. Each server's health score is sent to the OHS plug-in using a predefined HTTP response header, `X-WebLogic-Server-Health-Score`.

- Default Health Score Plug-In
  WebLogic Server provides a default plug-in implementation, which is used if a custom plug-in is not defined.
- Custom Plug-In Implementation
  WebLogic Server has defined the following interface that lets a service provider, define a custom plug-in implementation for calculating the health score of a clustered Managed Server instance.

Default Health Score Plug-In

WebLogic Server provides a default plug-in implementation, which is used if a custom plug-in is not defined.

The WebLogic Server provided plug-in calculates a server's health score based on:

- CPU load
- Heap usage
- Work Manager stuck threads count
- Data source pending connection request counts
- Health State

The default health score calculation will first check the server's health state. After this, an algorithm will consider a set of metrics. For each metric, the algorithm will calculate a "health score reduction" which is the amount that the health will be reduced because of this metric. The health score is just 100 minus the health score reduction. The metric causing the single largest health score reduction is used to arrive at the final health score, which will be 100 minus this health score reduction value.

Health State

- If the server health state is `HEALTH_FAILED`, then the health score is 0.
- If the server health state is `HEALTH_OVERLOADED`, then the health score reduction is 50.
- For any other health state, the health state value does not affect the health score.

> **Note:**
>
> The inclusion of `HEALTH_OVERLOADED` means that you can additionally configure health thresholds using the OverloadProtectionMBean.

Custom Plug-In Implementation

WebLogic Server has defined the following interface that lets a service provider, define a custom plug-in implementation for calculating the health score of a clustered Managed Server instance.

Use the Health Service Provider Interface (SPI) for calculating the health score for each Managed Server in a cluster. Using the health service provider interface, you can implement a custom algorithm, based on metrics related to the overall health of the Managed Server instance. The plug-in implementation should be configured on every clustered Managed Server instance for which OHS is interested in routing requests based on the health score.

```
/**
* Plugin interface for calculating health score.
*/
public interface HealthScore {
  int calculateHealthScore();
}
```

- Example Custom Health Score Plug-In Implementation
- Configuring the Health Score

Example Custom Health Score Plug-In Implementation

The following example plug-in implementation illustrates:

1. Looking up the reference to the local MBeanServer through JNDI.
2. Retrieving the relevant runtime MBeans.
3. Calculating a health score based on the relevant metrics.

```
package com.oracle.weblogic;

public class HealthScorePlugin implements HealthScore {

  // public default constructor needed for instantiation
  public HealthScorePlugin() { }

  @Override
  public int calculateHealthScore() {
    try {
      // Lookup local MBeanServer from JNDI.
      InitialContext ctx = new InitialContext();
      MBeanServer mbeanServer = (MBeanServer) ctx.lookup("java:comp/jmx/runtime");


      // Example of how to retrieve runtime mbeans.
      ObjectName objectName = new
ObjectName("com.bea:Name=RuntimeService,Type=weblogic.management.mbeanservers.runtime.Run
timeServiceMBean");
      ObjectName serverRuntime = (ObjectName) mbeanServer.getAttribute(objectName,
"ServerRuntime");
      ObjectName jvmRuntime = (ObjectName) mbeanServer.getAttribute(serverRuntime,
"JVMRuntime");
      ObjectName threadPoolRuntime = (ObjectName)
mbeanServer.getAttribute(serverRuntime, "ThreadPoolRuntime");
      ObjectName jdbcServiceRuntime = (ObjectName)
mbeanServer.getAttribute(serverRuntime, "JDBCServiceRuntime");

      return getHeapFreeHealthScore(mbeanServer, jvmRuntime);

    } catch (Exception e) {
      throw new RuntimeException(e);
    }
  }
```

```
  /**
   * Example of how to retrieve "HeapFreePercent" attribute from JVMRuntime MBean and
calculate related health score.
  */
  private int getHeapFreeHealthScore(MBeanServer mbeanServer, ObjectName jvmRuntime)
throws ReflectionException, AttributeNotFoundException,
        InstanceNotFoundException, MBeanException {

    int heapFreePercent = (int) mbeanServer.getAttribute(jvmRuntime, "HeapFreePercent");
    System.out.println("heapFreePercent: " + heapFreePercent);
    return 100 - heapFreePercent;
  }
}
```

Configuring the Health Score

> **Note:**
>
> The WebLogic Health Score Service is *disabled* by default.

The health score configuration and plug-in registration of service providers can be defined using a HealthScoreMBean at both the domain and server level.

- Configuring the health score at the domain level (using the DomainMBean) will apply to all Managed Servers defined in the WebLogic domain.
- You can configure individual health scores at the server level, which will override any domain level configuration.
- To make the service provider plug-in class implementation available, add it to the WebLogic Server system classpath.

See the following example health score configuration at WebLogic domain level.

```
<domain>
...
  <health-score>
    <enabled>true</enabled>
    <plugin-class-name>com.oracle.weblogic.HealthScorePlugin</plugin-class-name>
    <calculate-interval-secs>10</calculate-interval-secs>
  </health-score>
</domain>
```

WLST Example

The following WLST examples illustrate how to enable the health score service and how to define a health score configuration at the domain level. When configured at the domain level, the HealthScoreMBean configuration applies to all the Managed Servers in the domain.

This example enables the health score service.

```
# Connect to the AdminServer.
connect(adminUsername, adminPassword, adminURL)

edit()
startEdit()

# Navigate to Health Score at Domain.
cd('HealthScore/your_domain')
```

```
cmo.setEnabled(true)

save()
activate()

disconnect()
```

This example defines a health score with these custom values:

- `PluginClassName` - The class name of the custom health score plug-in to be instantiated and used by the health score service to calculate the health score of the server.
- `CalculateIntervalSecs` - The interval (time in seconds) at which WebLogic Server will call the health score plug-in for calculating the server's health score.

```
# Connect to the AdminServer.
connect(adminUsername, adminPassword, adminURL)

edit()
startEdit()

# Navigate to Health Score at Domain.
cd('HealthScore/your_domain')
cmo.setPluginClassName('weblogic.health.HealthScorePlugin')
com.setCalculateIntervalSecs(10)


save()
activate()

disconnect()
```

Setting Up HTTP Access Logs

WebLogic Server can keep a log of all HTTP transactions in a text file, in either *common log format* or *extended log format*.

Common log format is the default. Extended log format allows you to customize the information that is recorded. You can set the attributes that define the behavior of HTTP access logs for each server instance or for each virtual host that you define.

To set up HTTP logging for a server or a virtual host, in the Remote Console, in the **Edit Tree**:

- For servers, go to **Environment**: **Servers**: *myServer*. Enable and configure HTTP access logs on the **Logging**: **HTTP** subtab.
- For virtual hosts, go to **Environment**: **Virtual Hosts**: *myVirtualHost*. Specify the HTTP log file settings on the **Logging** page.

- Log Rotation
- Common Log Format
- Setting Up HTTP Access Logs by Using Extended Log Format

Log Rotation

You can rotate the log file based on either the size of the file or after a specified amount of time has passed. When either criterion is met, the current access log file is closed and a new access log file is started. If you do not configure log rotation, the HTTP access log file grows indefinitely. You can configure the name of the access log file to include a time and date stamp

that indicates when the file was rotated. If you do not configure a time stamp, each rotated file name includes a numeric portion that is incremented upon each rotation. Separate HTTP access logs are kept for each Virtual Host you have defined.

Common Log Format

The default format for logged HTTP information is the common log format. See http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format.

This standard format follows the pattern:

```
host RFC931 auth_user [day/month/year:hour:minute:second
  UTC_offset] "request" status bytes
```

where:

host
Either the DNS name or the IP number of the remote client

RFC931
Any information returned by IDENTD for the remote client; WebLogic Server does not support user identification

auth_user
If the remote client user sent a userid for authentication, the user name; otherwise "-"

day/month/year:hour:minute:second UTC_offset
Day, calendar month, year and time of day (24-hour format) with the hours difference between local time and GMT, enclosed in square brackets

"request"
First line of the HTTP request submitted by the remote client enclosed in double quotes

status
HTTP status code returned by the server, if available; otherwise "-"

bytes
Number of bytes listed as the content-length in the HTTP header, not including the HTTP header, if known; otherwise "-"

Setting Up HTTP Access Logs by Using Extended Log Format

WebLogic Server also supports extended log file format, version 1.0, an emerging standard defined by the draft specification from the W3C at http://www.w3.org/TR/WD-logfile.html. The current definitive reference is on the W3C Technical Reports and Publications page at http://www.w3.org/TR/.

The extended log format allows you to specify the type and order of information recorded about each HTTP communication. To enable this format in the WebLogic Remote Console:

1. In the **Edit Tree**, go to **Environment**: **Servers**: *myServer*.
2. Then, navigate to the **Logging**: **HTTP** subtab.
3. Make sure that **HTTP access log file enabled** option is on.
4. Click **Show Advanced Fields**.
5. In the field labeled **Format**, select **Extended**.

In the field labeled **Extended Logging Format Fields**, you can select one or more of the fields described in Supported Field Identifiers. If you want to add custom fields to an HTTP access log file, see Creating Custom Field Identifiers for details.

You specify the information that should be recorded in the log file with directives, included in the actual log file itself. A directive begins on a new line and starts with a pound sign (`#`). If the log file does not exist, a new log file is created with default directives. However, if the log file already exists when the server starts, it must contain valid directives at the head of the file.

- Creating the Fields Directive
- Supported Field Identifiers
- Creating Custom Field Identifiers

Creating the Fields Directive

The first line of your log file must contain a directive stating the version number of the log file format. You must also include a `Fields` directive near the beginning of the file:

```
#Version: 1.0
#Fields: xxxx xxxx xxxx ...
```

Where each `xxxx` describes the data fields to be recorded. Field types are specified as either simple identifiers, or may take a prefix-identifier format, as defined in the W3C specification. For example:

```
#Fields: date time cs-method cs-uri
```

This identifier instructs the server to record the date and time of the transaction, the request method that the client used, and the URI of the request for each HTTP access. Each field is separated by white space, and each record is written to a new line, appended to the log file.

> **Note:**
>
> The `#Fields` directive must be followed by a new line in the log file, so that the first log message is not appended to the same line.

Supported Field Identifiers

The following identifiers are supported, and do not require a prefix.

date
Date at which transaction completed, field has type `<date>`, as defined in the W3C specification.

time
Time at which transaction completed, field has type `<time>`, as defined in the W3C specification.

time-taken
Time taken for transaction to complete in seconds, field has type `<fixed>`, as defined in the W3C specification.

bytes
Number of bytes transferred, field has type `<integer>`.
Note that the cached field defined in the W3C specification is not supported in WebLogic Server.
The following identifiers require prefixes, and cannot be used alone. The supported prefix combinations are explained individually.

- IP Address Related Fields
- DNS Related Fields
- Diagnostic Message Correlation Fields

IP Address Related Fields

These fields give the IP address and port of either the requesting client, or the responding server. These fields have type <address>, as defined in the W3C specification. The supported fields are:

c-ip
The IP address of the client.

s-ip
The IP address of the server.

DNS Related Fields

These fields give the domain names of the client or the server and have type `<name>`, as defined in the W3C specification. The supported fields are:

c-dns
The domain name of the requesting client.

s-dns
The domain name of the requested server.

sc-status
Status code of the response, for example (404) indicating a "File not found" status. This field has type `<integer>`, as defined in the W3C specification.

sc-comment
The comment returned with status code, for instance "File not found". This field has type `<text>`.

cs-method
The request method, for example GET or POST. This field has type `<name>`, as defined in the W3C specification.

cs-uri
The full requested URI. This field has type `<uri>`, as defined in the W3C specification.

> **Note:**
>
> When extended log format is enabled, the logged URI is truncated if its length exceeds 256 characters, which is the default limit. You can increase the maximum URI length by specifying it in following argument to the command that starts WebLogic Server:
>
> `-Dweblogic.servlet.maxLoggingURILength=`*`length`*

cs-uri-stem
Only the stem portion of URI (omitting query). This field has type `<uri>`, as defined in the W3C specification.

cs-uri-query
Only the query portion of the URI. This field has type `<uri>`, as defined in the W3C specification.

Diagnostic Message Correlation Fields

These fields give message correlation information for diagnostic messages, helping you to determine relationships between messages across components. These fields are logged if the diagnostic context is present and populated for the executed request. The diagnostic context may be present if it is propagated into the server with the incoming request, or it may be created for the request by WebLogic Server if the diagnostic context is enabled. The supported fields are:

ctx-ecid
The Execution Context ID (ECID). The ECID is a globally unique identifier associated with the execution of a particular request.

ctx-rid
The Relationship ID (RID). The RID distinguishes the work done in one thread on one process, from work done by any other threads on this and other processes on behalf of the same request.

If the diagnostic context does not exist, or the values of the ECID and RID are not available in the diagnostic context, a hyphen (-) is logged as their values. For more information about the ECID and RID, see Correlating Messages Across Log Files and Components in *Administering Oracle Fusion Middleware*.

Creating Custom Field Identifiers

You can also create user-defined fields for inclusion in an HTTP access log file that uses the extended log format (ELF). To create a custom field, you identify the field in the ELF log file using the `Fields` directive and then you create a matching Java class that generates the desired output. You can create a separate Java class for each field, or the Java class can output multiple fields. For a sample of the Java source for such a class, see Example 2-1.

To create a custom field:

1. Include the field name in the `Fields` directive, using the form:

   `x-`*`myCustomField`*`.`

   Where *`myCustomField`* is a fully-qualified class name.

See Creating the Fields Directive.

2. Create a Java class with the same fully-qualified class name as the custom field you defined with the `Fields` directive (for example `myCustomField`). This class defines the information you want logged in your custom field. The Java class must implement the following interface:

   `weblogic.servlet.logging.CustomELFLogger`

   In your Java class, you must implement the `logField()` method, which takes a `HttpAccountingInfo` object and `FormatStringBuffer` object as its arguments:

   - Use the `HttpAccountingInfo` object to access HTTP request and response data that you can output in your custom field. Getter methods are provided to access this information. For a complete listing of these get methods, see Get Methods of the HttpAccountingInfo Object.
   - Use the `FormatStringBuffer` class to create the contents of your custom field. Methods are provided to create suitable output.

3. Compile the Java class and add the class to the `CLASSPATH` statement used to start WebLogic Server. You will probably need to modify the `CLASSPATH` statements in the scripts that you use to start WebLogic Server.

   > **Note:**
   >
   > Do not place this class inside of a Web application or Enterprise application in exploded or jar format.

4. Configure WebLogic Server to use the extended log format. See Setting Up HTTP Access Logs by Using Extended Log Format.

   > **Note:**
   >
   > When writing the Java class that defines your custom field, do not execute any code that is likely to slow down the system (For instance, accessing a DBMS or executing significant I/O or networking calls.) Remember, an HTTP access log file entry is created for *every* HTTP request.

   > **Note:**
   >
   > If you want to output more than one field, delimit the fields with a tab character. For more information on delimiting fields and other ELF formatting issues, see "Extended Log Format" at `http://www.w3.org/TR/WD-logfile-960221.html`.

- Get Methods of the HttpAccountingInfo Object

Get Methods of the HttpAccountingInfo Object

The following methods return various data regarding the HTTP request. These methods are similar to various methods of `javax.servlet.ServletRequest`, `javax.servlet.http.Http.ServletRequest`, and `javax.servlet.http.HttpServletResponse`.

The Javadoc for these interfaces is available at the following locations:

- https://javaee.github.io/javaee-spec/javadocs/javax/servlet/ServletRequest.html
- https://javaee.github.io/javaee-spec/javadocs/javax/servlet/ServletResponse.html
- https://javaee.github.io/javaee-spec/javadocs/javax/servlet/http/HttpServletRequest.html

For details about these methods, see the corresponding methods in the Java interfaces listed in the following table, or refer to the specific information contained in this table.

Table 2-2 Getter Methods of HttpAccountingInfo

HttpAccountingInfo Methods	Method Information
`Object getAttribute(String name);`	`javax.servlet.ServletRequest`
`Enumeration getAttributeNames();`	`javax.servlet.ServletRequest`
`String getCharacterEncoding();`	`javax.servlet.ServletRequest`
`int getResponseContentLength();`	`javax.servlet.ServletResponse.setContentLength()` This method gets the content length of the response, as set with the `setContentLength()` method.
`String getContentType();`	`javax.servlet.ServletRequest`
`Locale getLocale();`	`javax.servlet.ServletRequest`
`Enumeration getLocales();`	`javax.servlet.ServletRequest`
`String getParameter(String name);`	`javax.servlet.ServletRequest`
`Enumeration getParameterNames();`	`javax.servlet.ServletRequest`
`String[] getParameterValues(String name);`	`javax.servlet.ServletRequest`
`String getProtocol();`	`javax.servlet.ServletRequest`
`String getRemoteAddr();`	`javax.servlet.ServletRequest`
`String getRemoteHost();`	`javax.servlet.ServletRequest`
`String getScheme();`	`javax.servlet.ServletRequest`
`String getServerName();`	`javax.servlet.ServletRequest`
`int getServerPort();`	`javax.servlet.ServletRequest`
`boolean isSecure();`	`javax.servlet.ServletRequest`
`String getAuthType();`	`javax.servlet.http.HttpServletRequest`
`String getContextPath();`	`javax.servlet.http.HttpServletRequest`
`Cookie[] getCookies();`	`javax.servlet.http.HttpServletRequest`
`long getDateHeader(String name);`	`javax.servlet.http.HttpServletRequest`
`String getHeader(String name);`	`javax.servlet.http.HttpServletRequest`
`Enumeration getHeaderNames();`	`javax.servlet.http.HttpServletRequest`
`Enumeration getHeaders(String name);`	`javax.servlet.http.HttpServletRequest`
`int getIntHeader(String name);`	`javax.servlet.http.HttpServletRequest`
`String getMethod();`	`javax.servlet.http.HttpServletRequest`
`String getPathInfo();`	`javax.servlet.http.HttpServletRequest`
`String getPathTranslated();`	`javax.servlet.http.HttpServletRequest`
`String getQueryString();`	`javax.servlet.http.HttpServletRequest`
`String getRemoteUser();`	`javax.servlet.http.HttpServletRequest`
`String getRequestURI();`	`javax.servlet.http.HttpServletRequest`
`String getRequestedSessionId();`	`javax.servlet.http.HttpServletRequest`

Table 2-2 (Cont.) Getter Methods of HttpAccountingInfo

HttpAccountingInfo Methods	Method Information
`String getServletPath();`	`javax.servlet.http.HttpServletRequest`
`Principal getUserPrincipal();`	`javax.servlet.http.HttpServletRequest`
`boolean isRequestedSessionIdFromCookie();`	`javax.servlet.http.HttpServletRequest`
`boolean isRequestedSessionIdFromURL();`	`javax.servlet.http.HttpServletRequest`
`boolean isRequestedSessionIdFromUrl();`	`javax.servlet.http.HttpServletRequest`
`boolean isRequestedSessionIdValid();`	`javax.servlet.http.HttpServletRequest`
`byte[] getURIAsBytes();`	Returns the URI of the HTTP request as byte array. For example, If `GET /index.html HTTP/1.0` is the first line of an HTTP Request, `/index.html` is returned as an array of bytes.

`long getInvokeTime();`	Returns the starting time of `currentTimeMillis()`.
	To get the length of time taken by the servlet to send the response to the client, use the following code:
	`long milsec = System.currentTimeMillis() - metrics.getInvokeTime();`
	`Float sec = new Float(milsec / 1000.0);`

`int getResponseStatusCode();`	`javax.servlet.http.HttpServletResponse`
`String getResponseHeader(String name);`	`javax.servlet.http.HttpServletResponse`

Example 2-1 Java Class for Creating a Custom ELF Field

```
import weblogic.servlet.logging.CustomELFLogger;
import weblogic.servlet.logging.FormatStringBuffer;
import weblogic.servlet.logging.HttpAccountingInfo;
/* This example outputs the User-Agent field into a
 custom field called MyCustomField
*/
public class MyCustomField implements CustomELFLogger{
public void logField(HttpAccountingInfo metrics,
  FormatStringBuffer buff) {
  buff.appendValueOrDash(metrics.getHeader("User-Agent"));
  }
}
```

Preventing POST Denial-of-Service Attacks

A Denial-of-Service attack is a malicious attempt to overload a server with phony requests. One common type of attack is to send huge amounts of data in an HTTP POST method. You can set three attributes in WebLogic Server that help prevent this type of attack. These attributes are set in the Remote Console, under **Servers**: **Protocols**: **HTTP** or **Virtual Hosts**: **HTTP**. If you define these attributes for a virtual host, the values set for the virtual host override those set under Servers.

PostTimeout
Amount of time that WebLogic Server waits between receiving chunks of data in an HTTP POST.
The default value for `PostTimeout` is 30 seconds.

MaxPostTime
Maximum time that WebLogic Server spends receiving post data. If this limit is triggered, a `PostTimeoutException` is thrown and the following message is sent to the server log: `Post time exceeded MaxPostTime.`
The default value for `MaxPostTime` is 30 seconds.

MaxPostSize
Maximum number of bytes of data received from a single request. The configuration controls both POST and PUT requests. If the requested data exceeds the `MaxPostSize`, the system throws `MaxPostSizeExceeded` exception and sends the following message to the server log: `POST size exceeded the parameter MaxPostSize.`

- If the request contains chunked transfer encoding and the requested data exceeds the `MaxPostSize`, `MaxPostSizeException` is thrown.
- If the request comprises of content-length set and the requested data exceeds the `MaxPostSize`, the message `POST size exceeded the parameter MaxPostSize.` is written to server log and an HTTP error code 413 (Request Entity Too Large) is sent back to the client.

The default value for `MaxPostSize` is -1.

Setting Up WebLogic Server for HTTP Tunneling

HTTP tunneling provides a way to simulate a stateful socket connection between WebLogic Server and a Jakarta client when your only option is to use the HTTP protocol. It is generally used to *tunnel* through an HTTP port in a security firewall. HTTP is a stateless protocol, but WebLogic Server provides tunneling functionality to make the connection appear to be a regular T3Connection. However, you can expect some performance loss in comparison to a normal socket connection.

> **Note:**
>
> Oracle does not recommend enabling tunneling on channels that are available external to the firewall.

- Configuring the HTTP Tunneling Connection
- Connecting to WebLogic Server from the Client

Configuring the HTTP Tunneling Connection

Under the HTTP protocol, a client may only make a request, and then accept a reply from a server. The server may not voluntarily communicate with the client, and the protocol is stateless, meaning that a continuous two-way connection is not possible.

WebLogic HTTP tunneling simulates a T3Connection via the HTTP protocol, overcoming these limitations. There are attributes that you can configure in the WebLogic Remote Console to tune a tunneled connection for performance. It is advised that you leave them at their default settings unless you experience connection problems. These properties are used by the server to determine whether the client connection is still valid, or whether the client is still alive.

Enable Tunneling
Enables or disables HTTP tunneling. HTTP tunneling is disabled by default.

Note that the server must also support both the HTTP and T3 protocols in order to use HTTP tunneling.

Tunneling Client Ping
The interval (in seconds) at which to check a tunneled client to see if it is still alive. Default is 45 seconds; valid range is 20 to 900 seconds.

Tunneling Client Timeout
If the number of seconds set in this attribute have elapsed since the client last sent a request to the server (in response to a reply), then the server regards the client as dead, and terminates the HTTP tunnel connection. The server checks the elapsed time at the interval specified by this attribute, when it would otherwise respond to the client's request. Default is 40 seconds; valid range is 10 to 900 seconds.

Connecting to WebLogic Server from the Client

When your client requests a connection with WebLogic Server, all you need to do in order to use HTTP tunneling is specify the HTTP protocol in the URL. For example:

```
Hashtable env = new Hashtable();
env.put(Context.PROVIDER_URL, "http://wlhost:80");
Context ctx = new InitialContext(env);
```

On the client side, a special tag is appended to the `http` protocol, so that WebLogic Server knows this is a tunneling connection, instead of a regular HTTP request. Your application code does not need to do any extra work to make this happen.

The client must specify the port in the URL, even if the port is 80. You can set up your WebLogic Server instance to listen for HTTP requests on any port, although the most common choice is port 80 since requests to port 80 are customarily allowed through a firewall.

See Specify Listen Ports in the *Oracle WebLogic Remote Console Online Help*.

Using Native I/O for Serving Static Files (Windows Only)

When running WebLogic Server on Windows NT/2000/XP you can specify that WebLogic Server use the native operating system call `TransmitFile` instead of using Java methods to serve static files such as HTML files, text files, and image files. Using native I/O can provide performance improvements when serving larger static files.

To use native I/O, add two parameters to the `web.xml` deployment descriptor of a Web application containing the files to be served using native I/O. The first parameter, `weblogic.http.nativeIOEnabled` should be set to `TRUE` to enable native I/O file serving. The second parameter, `weblogic.http.minimumNativeFileSize` sets the minimum file size for using native I/O. If the file being served is larger than this value, native I/O is used. If you do not specify this parameter, a value of 4K is used by default.

Generally, native I/O provides greater performance gains when serving larger files. However, as the load on the machine running WebLogic Server increases, these gains diminish. You may need to experiment to find the correct value for `weblogic.http.minimumNativeFileSize`.

The following example shows the complete entries that should be added to the `web.xml` deployment descriptor. These entries must be placed in the `web.xml` file after the `<distributable>` element and before the `<servlet>` element.

```
<context-param>
  <param-name>weblogic.http.nativeIOEnabled</param-name>
  <param-value>TRUE</param-value>
```

```
</context-param>
<context-param>
  <param-name>weblogic.http.minimumNativeFileSize</param-name>
  <param-value>500</param-value>
</context-param>
```

`weblogic.http.nativeIOEnabled` can also be set as a context parameter in the FileServlet.

3

Using Work Managers to Optimize Scheduled Work

WebLogic Server helps you determine how your application prioritizes the execution of its work using a Work Manager. Based on rules you define and by monitoring actual runtime performance, WebLogic Server can optimize the performance of your application and maintain service-level agreements. You define the rules and constraints for your application by defining a Work Manger and applying it either globally to a WebLogic Server domain or to a specific application component.

- Understanding How WebLogic Server Uses Thread Pools
  WebLogic Server uses a thread pool to execute various types of work and prioritizes the execution of work based on the rules and the run-time metrics you define in the Work Manager.
- Understanding Work Managers
  WebLogic Server prioritizes work and allocates threads based on an execution model that takes into account administrator-defined parameters and actual run-time performance and throughput.
- Work Manager Scope
  Essentially, there are three types of Work Managers, each one characterized by its scope and how it is defined and used.
- Using Work Managers, Request Classes, and Constraints
  Work Managers, Request Classes, and Constraints require a definition and a mapping.
- Deployment Descriptor Examples
  Examine examples for defining Work Managers in various types of deployment descriptors.
- Work Managers and Execute Queues
  Learn how to enable backward compatibility with Execute Queues and how to migrate applications from using Execute Queues to Work Managers.
- Accessing Work Managers Using MBeans
- Using CommonJ With WebLogic Server
  WebLogic Server Work Managers provide server-level configuration that allows administrators a way to set dispatch-policies to their servlets and EJBs. WebLogic Server provides a programmatic way of handling work from within an application by implementing the `commonj.work` and `commonj.timers` packages of the CommonJ specification.

Understanding How WebLogic Server Uses Thread Pools

WebLogic Server uses a thread pool to execute various types of work and prioritizes the execution of work based on the rules and the run-time metrics you define in the Work Manager.

In previous versions of WebLogic Server, processing was performed in multiple execute queues. Different classes of work were executed in different queues, based on priority and ordering requirements, and to avoid deadlocks. In addition to the default execute queue, `weblogic.kernel.default`, there were pre-configured queues dedicated to internal administrative traffic, such as `weblogic.admin.HTTP` and `weblogic.admin.RMI`.

You could control thread usage by altering the number of threads in the default queue, or configure custom execute queues to ensure that particular applications had access to a fixed number of execute threads, regardless of overall system load.

Now WebLogic Server uses a single thread pool, in which all types of work are executed. WebLogic Server prioritizes work based on rules you define, and run-time metrics, including the actual time it takes to execute a request and the rate at which requests are entering and leaving the pool.

The common thread pool changes its size automatically to maximize throughput. The queue monitors throughput over time and based on history, determines whether to adjust the thread count. For example, if historical throughput statistics indicate that a higher thread count increased throughput, WebLogic increases the thread count. Similarly, if statistics indicate that fewer threads did not reduce throughput, WebLogic decreases the thread count. This new strategy makes it easier for administrators to allocate processing resources and manage performance, avoiding the effort and complexity involved in configuring, monitoring, and tuning custom executes queues.

Understanding Work Managers

WebLogic Server prioritizes work and allocates threads based on an execution model that takes into account administrator-defined parameters and actual run-time performance and throughput.

Administrators can configure a set of scheduling guidelines and associate them with one or more applications, or with particular application components. For example, you can associate one set of scheduling guidelines for one application, and another set of guidelines for other applications. At run time, WebLogic Server uses these guidelines to assign pending work and enqueued requests to execution threads.

> **Note:**
>
> Work requests from all Work Managers are executed by a single thread pool; separate thread pools are not created for each Work Manager.

To manage work in your applications, you define one or more of the following Work Manager components:

- Fair Share Request Class
- Response Time Request Class
- Min Threads Constraint
- Max Threads Constraint
- Capacity Constraint
- Context Request Class

See Request Classes or Constraints.

You can use any of these Work Manager components to control the performance of your application by referencing the name of the component in the application deployment descriptor. In addition, you may define a *Work Manager* that encapsulates all of the above components (except Context Request Class; see Example 3-3) and reference the name of the Work Manager in your application's deployment descriptor. You can define multiple Work Managers

—the appropriate number depends on how many distinct demand profiles exist across the applications you host on WebLogic Server.

You can configure Work Managers at the domain level, application level, and module level in one of the following configuration files, or by using the WebLogic Remote Console (see Scheduling Work in the *Oracle WebLogic Remote Console Online Help*):

- `config.xml`—Work Managers specified in `config.xml` can be assigned to any application, or application component, in the domain.
- `weblogic-application.xml`—Work Managers specified at the application level can be assigned to that application, or any component of that application.
- `weblogic-ejb-jar.xml` or `weblogic.xml`—Work Managers specified at the component level can be assigned to that component.
- `weblogic.xml`—Work Managers specified for a Web application.

Example 3-1 is an example of a Work Manager definition.

- Request Classes
- Constraints
- Stuck Thread Handling

Example 3-1 Work Manager Stanza

```
<work-manager>
<name>highpriority_workmanager</name>
   <fair-share-request-class>
      <name>high_priority</name>
      <fair-share>100</fair-share>
   </fair-share-request-class>
   <min-threads-constraint>
      <name>MinThreadsCountFive</name>
      <count>5</count>
   </min-threads-constraint>
</work-manager>
```

To assign the Work Manager in Example 3-1 to control the dispatch policy of the entire Web application, add the code in Example 3-2 to the Web application's `weblogic.xml` file:

Example 3-2 Referencing the Work Manager in a Web Application

```
<wl-dispatch-policy>highpriority-workmanager</wl-dispatch-policy>
```

To assign the Work Manager to control the dispatch policy of a particular servlet, add the following code to the Web application's `web.xml` file:

```
<servlet>
  ...
     <init-param>
        <param-name>wl-dispatch-policy</param-name>
        <param-value>highpriority_workmanager</param-value>
     </init-param>
  ...
</servlet>
```

The components you can define and use in a Work Manager are described in following sections:

- Request Classes
- Constraints

- Stuck Thread Handling
- Self-Tuning Thread Pool

Request Classes

A request class expresses a scheduling guideline that WebLogic Server uses to allocate threads to requests. Request classes help ensure that high priority work is scheduled before less important work, even if the high priority work is submitted after the lower priority work. WebLogic Server takes into account how long it takes for requests to each module to complete.

Request classes define a best effort. They do not guarantee that the configured ratio will be maintained consistently. The observed ratio may vary due to several factors during a period of sufficient demand, such as:

- The mixture of requests from different request classes in the queue at any particular time. For example, more requests than the configured ratio may be processed for a lower priority request class if there are not enough requests from a higher priority request class in the Work Manager queue.
- Because the ratio is specified in terms of thread-usage time, a larger number of shorter requests could be processed in the same amount of thread-usage time as a smaller number of time-consuming requests.

There are multiple types of request classes, each of which expresses a scheduling guideline in different terms. A Work Manager may specify only one request class.

- `fair-share-request-class`—Specifies the average thread-use time required to process requests. The default fair share value is 50.

  For example, assume that WebLogic Server is running two modules. The Work Manager for `ModuleA` specifies a `fair-share-request-class` of 80 and the Work Manager for `ModuleB` specifies a `fair-share-request-class` of 20.

  During a period of sufficient demand, with a steady stream of requests for each module such that the number requests exceed the number of threads, WebLogic Server will allocate 80% and 20% of the thread-usage time to `ModuleA` and `ModuleB`, respectively.

  > **Note:**
  >
  > The value of a fair share request class is specified as a relative value, not a percentage. Therefore, in the above example, if the request classes were defined as 400 and 100, they would still have the same relative values.

  A work manager will be assigned a fair share request class with a default fair share value of 50 if no request class is explicitly configured.

- `response-time-request-class`—Specifies a response time goal in milliseconds. Response time goals are not applied to individual requests. Instead, WebLogic Server computes a tolerable waiting time for requests with that class by subtracting the observed average thread use time from the response time goal, and schedules requests so that the average wait for requests with the class is proportional to its tolerable waiting time.

  There is no default response time value in response time requests classes. A response time goal must be specified for each response time request class.

For example, given that ModuleA and ModuleB in the previous example, have response time goals of 2000 ms and 5000 ms, respectively, and the actual thread use time for an individual request is less than its response time goal. During a period of sufficient demand, with a steady stream of requests for each module such that the number of requests exceed the number of threads, and no "think time" delays between response and request, WebLogic Server will schedule requests for `ModuleA` and `ModuleB` to keep the average response time in the ratio 2:5. The actual average response times for `ModuleA` and `ModuleB` might be higher or lower than the response time goals, but will be a common fraction or multiple of the stated goal. For example, if the average response time for `ModuleA` requests is 1,000 ms., the average response time for `ModuleB` requests is 2,500 ms.

The previous sections described request classes based on fair share and response time by relating the scheduling to other work using the same request class. A mix of fair share and response time request classes is scheduled with a marked bias in favor of response time scheduling.

The scheduling priorities of fair share and response time request classes are maintained separately. It it not possible to determine the relative priorities between a fair share request class and a response time request class. If it is important to maintain relative scheduling priorities of a set of work managers, they should all be configured with either response time request classes or fair share request classes.

- `context-request-class`—Assigns request classes to requests based on context information, such as the current user or the current user's group.

  For example, the `context-request-class` in Example 3-3 assigns a request class to requests based on the value of the request's `subject` and `role` properties.

  The `high_fairshare` and `low_fairshare` request classes referenced by the `context_workmanager` in Example 3-3 could be defined in the `config.xml` as follows:

```
<self-tuning>
  ...
    <fair-share-request-class>
      <name>high_fairshare</name>
      <target>myserver</target>
      <fair-share>75</fair-share>
    </fair-share-request-class>
    <fair-share-request-class>
      <name>low_fairshare</name>
      <target>myserver</target>
      <fair-share>25</fair-share>
    </fair-share-request-class>
 ...
</self-tuning>
```

> **Note:**
>
> If a Web application's Work Manager references a context request class, the first user call will go through the default request class; subsequent calls in same session will go through the user-defined request class.
>
> When using context request classes, set session timeout values to prevent sessions from expiring while requests wait in the Work Manager queue.

Example 3-3 Context Request Class

```
<work-manager>
  <name>context_workmanager</name>
```

```
  <context-request-class>
    <name>test_context</name>
    <context-case>
    <user-name>system</user-name>
    <request-class-name>high_fairshare</request-class-name>
  </context-case>
  <context-case>
    <group-name>everyone</group-name>
    <request-class-name>low_fairshare</request-class-name>
  </context-case>
  </context-request-class>
</work-manager>
```

Constraints

A constraint defines minimum and maximum numbers of threads allocated to execute requests and the total number of requests that can be queued or executing before WebLogic Server begins rejecting requests.

You can define the following types of constraints:

- `max-threads-constraint`—Limits the number of concurrent threads executing requests from the constrained work set. The default is unlimited. For example, consider a constraint defined with maximum threads of 10 and shared by 3 entry points. The scheduling logic ensures that not more than 10 threads are executing requests from the three entry points combined.

  You can define a `max-threads-constraint` in terms of a the availability of the resource that requests depend upon, such as a connection pool.

  A `max-threads-constraint` might, but does not necessarily, prevent a request class from taking its fair share of threads or meeting its response time goal. Once the constraint is reached the server does not schedule requests of this type until the number of concurrent executions falls below the limit. The server then schedules work based on the fair share or response time goal.

- `min-threads-constraint`—Guarantees the number of threads the server will allocate to affected requests to avoid deadlocks. The default is zero. A `min-threads-constraint` value of one is useful, for example, for a replication update request, which is called synchronously from a peer.

  A `min-threads-constraint` might not necessarily increase a fair share. This type of constraint has an effect primarily when the server instance is close to a deadlock condition. In that case, the constraint will cause WebLogic Server to schedule a request even if requests in the service class have gotten more than its fair share recently.

- `capacity`—Causes the server to reject requests only when it has reached its capacity. The default is -1. Note that the capacity includes all requests, queued or executing, from the constrained work set. Work is rejected either when an individual capacity threshold is exceeded or if the global capacity is exceeded. This constraint is independent of the global queue threshold.

  Note that the `capacity` constraint is not enforced if the request is made by a user belonging to the WebLogic Server Administrators group.

Stuck Thread Handling

In response to stuck threads, you can define a Stuck Thread Work Manager component that can shut down the Work Manager, move the application into admin mode, or mark the server instance as failed.

For example, the Work Manager defined in Example 3-4 shuts down the Work Manager when two threads are stuck for longer than 30 seconds.

Example 3-4 Stuck-Thread Work Manager

```
<work-manager>
   <name>stuckthread_workmanager</name>
   <work-manager-shutdown-trigger>
      <max-stuck-thread-time>30</max-stuck-thread-time>
      <stuck-thread-count>2</stuck-thread-count>
   </work-manager-shutdown-trigger>
</work-manager>
```

Self-Tuning Thread Pool

WebLogic Server maintains three groups of threads for the self-tuning thread pool:

- Running threads: threads that are currently executing work requests submitted to Work Managers
- Idle threads: threads that are idly waiting for a work request

  Idle threads include threads that have completed their previous work requests and are waiting for new requests, as well as threads that are created by the self-tuning thread pool based on usage statistics in order to anticipate future workload.
- Standby threads: threads that are not currently processing or waiting for work requests

  Standby threads do not count toward the self-tuning thread pool thread count. When the self-tuning thread pool decides to decrease the thread count based on usage statistics, threads are moved from the group of idle threads into the group of standby threads. Conversely, when the self-tuning thread pool decides to increase the thread count, it first tries to find threads in the standby thread group to move to the idle thread group. The self-tuning thread pool only creates new threads if there are not enough threads in the standby group.

  Threads are shut down when the number of standby threads reaches an internal maximum limit of 256. Ideally, a number of standby threads are ready if WebLogic Server needs to increase the self-tuning thread pool count occurs so that the WebLogic Server instance can avoid creating new threads at a time when workload is high. Standby threads can also be created and used to satisfy minimum threads constraints. See Constraints.

- Self-Tuning Thread Pool Size
- ThreadLocal Clean Out

Self-Tuning Thread Pool Size

By default, the self-tuning thread pool size limit is 400. This limit includes all running and idle threads, but does not include any standby threads. You can configure the limit using the `SelfTuningThreadPoolSizeMax` attribute in the `KernelMBean`. You may choose a higher size limit if your system can support additional workload even when the self-tuning thread pool has reached its upper thread count limit. Contrarily, you may choose to lower the limit if your system resources, such as CPU, become overloaded at a lower thread count. However, if lowering the `SelfTuningThreadPoolSizeMax` limit, note that if the value is set too low, the self-tuning thread pool may not be allowed to create enough threads to handle the system workload. This could result in a backlog of pending work requests on some Work Managers.

> **Note:**
>
> Minimum threads constraints can affect the number of threads that are executing work requests for Work Managers, especially if the WebLogic Server instance is under heavy load.

The self-tuning thread pool does not consider the `SelfTuningThreadPoolSizeMax` attribute when creating a new standby thread to process incoming work requests for a Work Manager to satisfy its allocated minimum threads constraint. This is due to the importance of allocating threads for processing work requests for Work Managers with minimum threads constraints, which are designed to be used to avoid server-to-server deadlocks.

As a result, the maximum possible number of threads maintained by the self-tuning thread pool is the sum of the configured `SelfTuningThreadPoolSizeMax` attribute value and the sum of the values for all minimum threads constraints configured in the WebLogic Server instance, assuming a worst-case scenario where the configured number of threads are allocated to all configured minimum threads constraints.

ThreadLocal Clean Out

To clean up stray ThreadLocal use by applications and third-party libraries, configure the `eagerThreadLocalCleanup` attribute in the `KernelMBean`. The `eagerThreadLocalCleanup` attribute specifies whether to clean up all ThreadLocal storage from self-tuning thread pools after they have finished processing each work request.

By default, the `eagerThreadLocalCleanup` attribute is set to false, in which the self-tuning thread pool only cleans up ThreadLocal storage when a thread returns to a standby pool and after an application is undeployed.

Setting the `eagerThreadLocalCleanup` attribute to true ensures that all thread pool threads have no leftover ThreadLocal values from previous requests when running work for a new request. However, overhead occurs from cleaning up ThreadLocal storage after each work request and then reestablishing ThreadLocal values for each new request. Since some applications cache objects that are expensive to create in the ThreadLocal storage, cleaning up ThreadLocal values after each request may negatively impact performance on those applications.

Work Manager Scope

Essentially, there are three types of Work Managers, each one characterized by its scope and how it is defined and used.

- The Default Work Manager
- Global Work Managers
- Application-scoped Work Managers

The Default Work Manager

To handle thread management and perform self-tuning, WebLogic Server implements a default Work Manager. This Work Manager is used by an application when no other Work Managers are specified in the application's deployment descriptors.

In many situations, the default Work Manager may be sufficient for most application requirements. WebLogic Server thread-handling algorithms assign each application its own fair share by default. Applications are given equal priority for threads and are prevented from monopolizing them.

Overriding the Default Work Manager

You can override the behavior of the default Work Manager by creating and configuring a global Work Manager called `default`. This allows you to control the default thread-handling behavior of WebLogic Server.

> **Note:**
>
> When you override the default Work Manager, all instances are overridden.

When to Use Work Managers

Use the following guidelines to determine when you might want to use Work Managers to customize thread management:

> **Note:**
>
> To use Work Manager, it is mandatory to meet one of the guidelines.

- The default fair share (50) is not sufficient.

  This usually occurs in situations where one application needs to be given a higher priority over another.
- A response time goal is required.
- A minimum thread constraint needs to be specified to avoid server deadlock

Global Work Managers

You can create global Work Managers that are available to all applications and modules deployed on a server, in the WebLogic Remote Console and in `config.xml`.

An application uses a globally-defined Work Manager as a template. Each application creates its own instance which handles the work associated with that application and separates that work from other applications. This separation is used to handle traffic directed to two applications which are using the same dispatch policy. Handling each application's work separately, allows an application to be shut down without affecting the thread management of another application. Although each application implements its own Work Manager instance, the underlying components are shared.

Application-scoped Work Managers

In addition to globally-scoped Work Managers, you can also create Work Managers that are available only to a specific application or module. You can define application-scoped Work Managers in the WebLogic Remote Console and in the following descriptors:

- `weblogic-application.xml`
- `weblogic-ejb-jar.xml`
- `weblogic.xml`

If you do not explicitly assign a Work Manager to an application, it uses the default Work Manager.

A method is assigned to a Work Manager, using the <dispatch-policy> element in the deployment descriptor. The <dispatch-policy> can also identify a custom execute queue, for backward compatibility. For an example, see Example 3-2.

Using Work Managers, Request Classes, and Constraints

Work Managers, Request Classes, and Constraints require a definition and a mapping.

- A definition. You may define Work Managers, Request Classes, or Constraints globally in the domain's configuration using the WebLogic Remote Console (in the **Edit Tree**, see **Scheduling**: **Work Managers**) or you may define them in one of the deployment descriptors listed previously. In either case, you assign a name to each.
- A mapping. In your deployment descriptors you reference one of the Work Managers, Request Classes, or Constraints by its name.
- Dispatch Policy for EJB
- Dispatch Policy for Web Applications

Dispatch Policy for EJB

`weblogic-ejb-jar.xml`—The value of the existing `dispatch-policy` tag under `weblogic-enterprise-bean` can be a named `dispatch-policy`. For backwards compatibility, it can also name an ExecuteQueue. In addition, Oracle allows `dispatch-policy`, `max-threads`, and `min-threads`, to specify named (or unnamed with a numeric value for constraints) policy and constraints for a list of methods, analogously to the present `isolation-level` tag.

Dispatch Policy for Web Applications

`weblogic.xml`—Also supports mappings analogous to the `filter-mapping` of the `web.xml`, where named dispatch-policy, max-threads, or min-threads are mapped for url-patterns or servlet names.

Deployment Descriptor Examples

Examine examples for defining Work Managers in various types of deployment descriptors.

For additional and detailed reference, see the schema for these deployment descriptors:

- weblogic-ejb-jar.xml schema: http://xmlns.oracle.com/weblogic/weblogic-ejb-jar/1.7/weblogic-ejb-jar.xsd
- weblogic-application.xml schema: http://xmlns.oracle.com/weblogic/weblogic-application/1.8/weblogic-application.xsd
- weblogic.xml schema: See weblogic.xml Deployment Descriptor Elements in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

Example 3-5 weblogic-ejb-jar.xml With Work Manager Entries

```
<weblogic-ejb-jar xmlns="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar"
   xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar
   http://xmlns.oracle.com/weblogic/weblogic-ejb-jar/1.7/weblogic-ejb-jar.xsd">

<weblogic-enterprise-bean>
   <ejb-name>WorkEJB</ejb-name>
   <jndi-name>core_work_ejb_workbean_WorkEJB</jndi-name>
   <dispatch-policy>weblogic.kernel.System</dispatch-policy>
</weblogic-enterprise-bean>

<weblogic-enterprise-bean>
   <ejb-name>NonSystemWorkEJB</ejb-name>
   <jndi-name>core_work_ejb_workbean_NonSystemWorkEJB</jndi-name>
   <dispatch-policy>workbean_workmanager</dispatch-policy>
</weblogic-enterprise-bean>

<weblogic-enterprise-bean>
   <ejb-name>MinThreadsWorkEJB</ejb-name>
   <jndi-name>core_work_ejb_workbean_MinThreadsWorkEJB</jndi-name>
   <dispatch-policy>MinThreadsCountFive</dispatch-policy>
</weblogic-enterprise-bean>

<work-manager>
   <name>workbean_workmanager</name>
</work-manager>

<work-manager>
   <name>stuckthread_workmanager</name>
   <work-manager-shutdown-trigger>
      <max-stuck-thread-time>30</max-stuck-thread-time>
      <stuck-thread-count>2</stuck-thread-count>
   </work-manager-shutdown-trigger>
</work-manager>

<work-manager>
   <name>minthreads_workmanager</name>
   <min-threads-constraint>
      <name>MinThreadsCountFive</name>
      <count>5</count>
   </min-threads-constraint>
</work-manager>

<work-manager>
   <name>lowpriority_workmanager</name>
   <fair-share-request-class>
      <name>low_priority</name>
      <fair-share>10</fair-share>
   </fair-share-request-class>
</work-manager>
```

```
<work-manager>
<name>highpriority_workmanager</name>
   <fair-share-request-class>
      <name>high_priority</name>
      <fair-share>100</fair-share>
   </fair-share-request-class>
</work-manager>

<work-manager>
<name>veryhighpriority_workmanager</name>
   <fair-share-request-class>
      <name>veryhigh_priority</name>
      <fair-share>1000</fair-share>
   </fair-share-request-class>
</work-manager>
```

The EJBs in Example 3-6 are configured to get as many threads as there are instances of a resource they depend upon—a connection pool, and an application-scoped connection pool.

Example 3-6 weblogic-ejb-jar.xml with Connection Pool Based Max Thread Constraint

```
<weblogic-ejb-jar xmlns="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar"
   xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar
   http://xmlns.oracle.com/weblogic/weblogic-ejb-jar/1.7/weblogic-ejb-jar.xsd">

   <weblogic-enterprise-bean>
     <ejb-name>ResourceConstraintEJB</ejb-name>
      <jndi-name>core_work_ejb_resource_ResourceConstraintEJB</jndi-name>
      <dispatch-policy>test_resource</dispatch-policy>
   </weblogic-enterprise-bean>

   <weblogic-enterprise-bean>
      <ejb-name>AppScopedResourceConstraintEJB</ejb-name>
      <jndi-name>core_work_ejb_resource_AppScopedResourceConstraintEJB
      </jndi-name>
      <dispatch-policy>test_appscoped_resource</dispatch-policy>
   </weblogic-enterprise-bean>

<work-manager>
   <name>test_resource</name>
    <max-threads-constraint>
      <name>pool_constraint</name>
      <pool-name>testPool</pool-name>
   </max-threads-constraint>
</work-manager>

<work-manager>
   <name>test_appscoped_resource</name>
   <max-threads-constraint>
      <name>appscoped_pool_constraint</name>
      <pool-name>AppScopedDataSource</pool-name>
   </max-threads-constraint>
</work-manager>
</weblogic-ejb-jar>
```

Example 3-7 weblogic-ejb-jar.xml with commonJ Work Managers

For information using commonJ, see Using CommonJ With WebLogic Server and the commonJ Javadocs.

Example 3-8 weblogic-application.xml

```
<weblogic-application xmlns="http://xmlns.oracle.com/weblogic/weblogic-application"
   xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-application
   http://xmlns.oracle.com/weblogic/weblogic-web-app/1.9/weblogic-web-app.xsd">

   <max-threads-constraint>
      <name>j2ee_maxthreads</name>
      <count>1</count>
   </max-threads-constraint>

   <min-threads-constraint>
      <name>j2ee_minthreads</name>
      count>1</count>
   </min-threads-constraint>

   <work-manager>
      <name>J2EEScopedWorkManager</name>
      </work-manager>
</weblogic-application>
```

The Web application in Example 3-9 is deployed as part of the Enterprise application defined in Example 3-8. This Web application's descriptor defines two Work Managers. Both Work Managers point to the same max threads constraint, `j2ee_maxthreads`, which is defined in the application's `weblogic-application.xml` file. Each Work Manager specifies a different response time request class.

Example 3-9 Web Application Descriptor

```
<weblogic xmlns="http://xmlns.oracle.com/weblogic"
   xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.oracle.com/weblogic
   http://xmlns.oracle.com/weblogic/1.0/weblogic.xsd">

   <work-manager>
      <name>fast_response_time</name>
      <response-time-request-class>
         <name>fast_response_time</name>
         <goal-ms>2000</goal-ms>
      </response-time-request-class>
      <max-threads-constraint-name>j2ee_maxthreads
      </max-threads-constraint-name>
   </work-manager>

   <work-manager>
      <name>slow_response_time</name>
      <max-threads-constraint-name>j2ee_maxthreads
      </max-threads-constraint-name
      <response-time-request-class>
         <name>slow_response_time</name>
         <goal-ms>5000</goal-ms>
      </response-time-request-class>
   </work-manager>

</weblogic>
```

The descriptor in Example 3-10 defines a Work Manager using the context-request-class.

Example 3-10 Web Application Descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-web-app xmlns="http://xmlns.oracle.com/weblogic/weblogic-web-app"
xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-web-app
http://xmlns.oracle.com/weblogic/weblogic-web-app/1.9/weblogic-web-app.xsd">
 <work-manager>
  <name>foo-servlet-1</name>
  <request-class-name>test-fairshare2</request-class-name>
  <max-threads-constraint>
    <name>foo-mtc</name>
    <pool-name>oraclePool</pool-name>
  </max-threads-constraint>
 </work-manager>

 <work-manager>
  <name>foo-servlet</name>
  <context-request-class>
  <name>test-context</name>
  <context-case>
    <user-name>anonymous</user-name>
    <request-class-name>test-fairshare1</request-class-name>
  </context-case>

   <context-case>
    <group-name>everyone</group-name>
   </context-request-class>
 </work-manager>
</weblogic-web-app>
```

Work Managers and Execute Queues

Learn how to enable backward compatibility with Execute Queues and how to migrate applications from using Execute Queues to Work Managers.

- Enabling Execute Queues
- Migrating from Execute Queues to Work Managers

Enabling Execute Queues

WebLogic Server, Version 8.1, implemented Execute Queues to handle thread management in which you created thread-pools to determine how workload was handled. WebLogic Server still provides Execute Queues for backward compatibility, primarily to facilitate application migration. However, when developing new applications, you should use Work Managers to perform thread management more efficiently.

You can enable Execute Queues in the following ways:

- Using the command line option `-Dweblogic.Use81StyleExecuteQueues=true`
- Setting the `Use81StyleExecuteQueues` property via the Kernel MBean in `config.xml`.

Enabling Execute Queues disables all Work Manager configuration and thread self tuning. Execute Queues behave exactly as they did in WebLogic Server 8.1.

When enabled, Work Managers are converted to Execute Queues based on the following rules:

- If the Work Manager implements a minimum or maximum threads constraint, then an Execute Queue is created with the same name as the Work Manager. The thread count of the Execute Queue is based on the value defined in the constraint.
- If the Work Manager does not implement any constraints, the global default Execute Queue is used.

Migrating from Execute Queues to Work Managers

When an application is migrated from WebLogic Server 8.1, any Execute Queues defined in the server configuration before migration will still be present. WebLogic Server does not automatically convert the Execute Queues to Work Managers.

When an 8.1 application implementing Execute Queues is deployed on WebLogic Server 9.x, the Execute Queues are created to handle thread management for requests. However, only those requests whose dispatch-policy maps to an Execute Queue will take advantage of this feature.

Accessing Work Managers Using MBeans

Work Managers can be accessed using the WorkManagerMBean configuration MBean. WorkManagerMBean is accessed in the runtime tree or configuration tree depending on how the Work Manager is accessed by an application.

- If the Work Manager is defined at the module level, the WorkManagerRuntime MBean is available through the corresponding ComponentRuntimeMBean.
- If a Work Manager is defined at the application level, then WorkManagerRuntime is available through ApplicationRuntime.
- If a Work Manager is defined globally in `config.xml`, each application creates its own instance of the Work Manager. Each application has its own corresponding WorkManagerRuntime available at the application level.

See WorkManagerMBean.

Using CommonJ With WebLogic Server

WebLogic Server Work Managers provide server-level configuration that allows administrators a way to set dispatch-policies to their servlets and EJBs. WebLogic Server provides a programmatic way of handling work from within an application by implementing the `commonj.work` and `commonj.timers` packages of the CommonJ specification.

For specific information on the WebLogic Server implementation of CommonJ, see the CommonJ Javadocs.

The WebLogic Server implementation of CommonJ enables an application to break a single request task into multiple work items, and assign those work items to execute concurrently using multiple Work Managers configured in WebLogic Server. Applications that do not need to execute concurrent work items can also use configured Work Managers by referencing or creating Work Managers in their deployment descriptors or, for Jakarta Connectors, using the JCA API.

The following are some differences between the WebLogic Server implementation and the CommonJ specification:

- The RemoteWorkItem interface is an optional interface provided by the CommonJ specification and is not supported in WebLogic Server. WebLogic Server implements its

own cluster load balancing and failover policies. Workload management is based on these policies.

- WebLogic CommonJ timers behave differently than `java.util.Timer`. When the execution is greater that twice the period, the WebLogic CommonJ timer will skip some periods to avoid falling further behind. The `java.util.Timer` does not do this.
- In a WebLogic Server environment, the `WorkListener.WorkRejected` method is called when a thread becomes stuck.
- Accessing CommonJ Work Managers
- Mapping CommonJ to WebLogic Server Work Managers

Accessing CommonJ Work Managers

Unlike WebLogic Server Work Managers, which can only be accessed from an application via dispatch policies, you can access CommonJ Work Managers directly from an application. The following code example demonstrates how to lookup a CommonJ Work Manager using JNDI:

```
InitialContext ic = new InitialContext();
commonj.work.WorkManager wm =
(commonj.work.WorkManager)ic.lookup("java:comp/env/wm/myWM");
```

See CommonJ Javadocs.

Mapping CommonJ to WebLogic Server Work Managers

You can map an externally defined CommonJ Work Manager to a WebLogic Server Work Manager. For example, if you have a CommonJ Work Manager defined in a descriptor, `ejb-jar.xml`, for example, as:

```
<resource-ref>
   <res-ref-name>minthreads_workmanager</res-ref-name>
   <res-type>commonj.work.WorkManager</res-type>
   <res-auth>Container</res-auth>
   <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

You can link this to a WebLogic Server Work Manager by ensuring that the `name` element is identical in the WebLogic Server descriptor such as `weblogic-ejb-jar.xml`:

```
<work-manager>
   <name>minthreads_workmanager</name>
   <min-threads-constraint>
   <count>5</count>
   </min-threads-constraint>
</work-manager>
```

This procedure is similar for a resource-ref defined in `web.xml`. The WebLogic Server Work Manager can be defined in either a module descriptor (`weblogic-ejb-jar.xml` or `weblogic.xml`, for example) or in the application descriptor (`weblogic-application.xml`).

4

Avoiding and Managing Overload

Oracle WebLogic Server has overload protection features that help to detect, avoid, and recover from overload conditions. These features prevent the negative consequences that result from continuing to accept requests when the system capacity is reached. These consequences degrade the application performance and stability.

- Configuring WebLogic Server to Avoid Overload Conditions
  When system capacity is reached, if an application server continues to accept requests, application performance and stability can deteriorate.
- WebLogic Server Self-Monitoring
  WebLogic Server self-monitoring features aid in determining and reporting overload conditions.
- WebLogic Server Exit Codes
  When WebLogic Server exits it returns an exit code. The exit codes can be used by shell scripts or HA agents to decide whether a server restart is necessary.

Configuring WebLogic Server to Avoid Overload Conditions

When system capacity is reached, if an application server continues to accept requests, application performance and stability can deteriorate.

The following sections demonstrate how you can configure WebLogic Server to minimize the negative results of system overload.

- Limiting Requests in the Thread Pool
- Limiting HTTP Sessions
- Exit on Out of Memory Exceptions
- Stuck Thread Handling

Limiting Requests in the Thread Pool

In WebLogic Server, all requests, whether related to system administration or application activity—are processed by a single thread pool. An administrator can throttle the thread pool by defining a maximum queue length. Beyond the configured value, WebLogic Server will refuse requests, except for requests on administration channels.

> **Note:**
>
> Administration channels allow access only to administrators. The limit you set on the execute length does not effect administration channel requests, to ensure that reaching the maximum thread pool length does not prevent administrator access to the system. To limit the number of administration requests allowed in the thread pool, you can configure an administration channel, and set the MaxConnectedClients attribute for the channel.

When the maximum number of enqueued requests is reached, WebLogic Server immediately starts rejecting:

- Web application requests.
- Non-transactional RMI requests with a low fair share, beginning with those with the lowest fair share.

  If the overload condition continues to persist, higher priority requests will start getting rejected, with the exception of JMS and transaction-related requests, for which overload management is provided by the JMS and the transaction manager.

Throttle the thread pool by setting the **Shared Capacity For Work Managers** field in the WebLogic Remote Console (on the **Environments**: **Servers**: *myServer*: **Advanced**: **Overload** page). The default value of this field is 65536.

- Work Managers and Thread Pool Throttling

Work Managers and Thread Pool Throttling

An administrator can configure Work Managers to manage the thread pool at a more granular level, for sets of requests that have similar performance, availability, or reliability requirements. A Work Manager can specify the maximum requests of a particular request class that can be queued. The maximum requests defined in a Work Manager works with the global thread pool value. The limit that is reached first is honored.

See Using Work Managers to Optimize Scheduled Work.

Limiting HTTP Sessions

An administrator can limit the number of active HTTP sessions based on detection of a low memory condition. This is useful in avoiding out of memory exceptions.

WebLogic Server refuses requests that create new HTTP sessions after the configured threshold has been reached. In a WebLogic Server cluster, the proxy plug-in redirects a refused request to another Managed Server in the cluster. A non-clustered server instance can redirect requests to alternative server instance.

The Servlet container takes one of the following actions when maximum number of sessions is reached:

- If the server instance is in a cluster, the servlet container throws a `SessionCreationException`. Your application code should handle this run-time exception and send a relevant response.

  To implement overload protection, you should handle this exception and send a 503 response explicitly. This response can then be handled by the proxy or load balancer.

You set a limit for the number of simultaneous HTTP sessions in the deployment descriptor for the Web application. For example, the following element sets a limit of 12 sessions:

```
<session-descriptor>
   <max-in-memory-sessions>12</max-in-memory-sessions>
</session-descriptor>
```

Exit on Out of Memory Exceptions

Administrators can configure WebLogic Server to exit upon an out of memory exception. This feature allows you to minimize the impact of the out of memory condition—automatic shutdown

helps avoid application instability, and you can configure Node Manager or another high availability (HA) tool to automatically restart WebLogic Server, minimizing down-time.

You can configure this using the WebLogic Remote Console or by editing the following elements in the `config.xml` file:

```
<overload-protection>
   <panic-action>system-exit</panic-action>
</overload-protection>
```

See the description of the OverloadProtectionMBean in the *MBean Reference for Oracle WebLogic Server*.

Stuck Thread Handling

WebLogic Server checks for stuck threads periodically. If all application threads are stuck, a server instance marks itself failed, if configured to do so, exits. You can configure Node Manager or a third-party high-availability solution to restart the server instance for automatic failure recovery.

You can configure these actions to occur when not all threads are stuck, but the number of stuck threads have exceeded a configured threshold:

- Shut down the Work Manager if it has stuck threads. A Work Manager that is shut down will refuse new work and reject existing work in the queue by sending a rejection message. In a cluster, clustered clients will fail over to another cluster member.
- Shut down the application if there are stuck threads in the application. The application is shutdown by bringing it into admin mode. All Work Managers belonging to the application are shut down, and behave as described above. Once the stuck thread condition is cleared, the application automatically returns to running mode.
- Mark the server instance as failed and shut it down it down if there are stuck threads in the server. In a cluster, clustered clients that are connected or attempting to connect will fail over to another cluster member.

See the description of the OverloadProtectionMBean in the *MBean Reference for Oracle WebLogic Server*.

WebLogic Server Self-Monitoring

WebLogic Server self-monitoring features aid in determining and reporting overload conditions.

Overloaded Health State

WebLogic Server has a health state—`OVERLOADED`—which is returned by the `ServerRuntimeMBean.getHealthState()` when a server instance whose life cycle state is `RUNNING` becomes overloaded. This condition occurs as a result of low memory.

Upon entering the `OVERLOADED` state, server instances start rejecting requests from the Work Manager queue (if a Work Manager is configured), HTTP requests return a 503 Error (Service Unavailable), and RMI requests fail over to another server if clustered, otherwise, a remote exception is returned to the client.

The server instances health state returns to `OK` after the overload condition passes. An administrator can suspend or shut down an `OVERLOADED` server instance.

WebLogic Server Exit Codes

When WebLogic Server exits it returns an exit code. The exit codes can be used by shell scripts or HA agents to decide whether a server restart is necessary.

See WebLogic Server Exit Codes and Restarting After Failure in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

5

Configuring Concurrent Managed Objects

Learn about the Concurrent Managed Objects (CMOs) implemented by Oracle WebLogic Server to provide support for defining and implementing the Jakarta Concurrency.

- About Jakarta Concurrency Utilities
  The Jakarta Concurrency Utilities implements a standard API for providing asynchronous capabilities to Jakarta EE application components such as servlets and EJBs.
- How Concurrent Managed Objects Provide Concurrency for WebLogic Server Containers
  Learn how WebLogic Server provides concurrency capabilities to Jakarta EE applications by associating the Concurrency Utilities API with the Work Manager to make threads container-managed.
- Default Jakarta Concurrency Objects
  The Jakarta standard specifies that certain default resources be made available to applications, and defines specific JNDI names for these default resources. WebLogic Server makes these names available through the use of logical JNDI names, which map Jakarta standard JNDI names to specific WebLogic Server resources.
- Customized CMOs in Configuration Files
  You can define the customized CMOs at the application and module level, or referenced from an application component environment (ENC) that is bound to JNDI.
- Global CMO Templates
  In addition to the JSR 236 default CMOs, you can also define global CMOs as templates in the domain's configuration by using the WebLogic Remote Console and the configuration MBeans.
- Configuring Concurrent Constraints
  Constraints can also be defined globally in the domain's configuration using the WebLogic Remote Console and configuration MBeans. Concurrent constraints specified in the `config.xml` can be assigned to any application or application component in the domain.
- Querying CMOs
  You can query global CMOs using administrative tools such as the Remote Console and MBeans:

About Jakarta Concurrency Utilities

The Jakarta Concurrency Utilities implements a standard API for providing asynchronous capabilities to Jakarta EE application components such as servlets and EJBs.

As described in the *The Java EE 8 Tutorial*, the two main concurrency concepts are processes and threads:

- Processes are primarily associated with applications running on the operating system (OS). A process has specific runtime resources to interact with the underlying OS and allocate other resources, such as its own memory, just as the JVM process does.
- Threads share some features with processes, because both consume resources from the OS or the execution environment. But threads are easier to create and consume fewer resources than a process.

The primary components of the concurrency utilities are:

- **ManagedExecutorService (MES):** Used by applications to execute submitted tasks asynchronously. Tasks are executed on threads that are started and managed by the container. The context of the container is propagated to the thread executing the task.
- **ManagedScheduledExecutorService (MSES):** Used by applications to execute submitted tasks asynchronously at specific times. Tasks are executed on threads that are started and managed by the container. The context of the container is propagated to the thread executing the task.
- **ManagedThreadFactory (MTF):** Used by applications to create managed threads. The threads are started and managed by the container. The context of the container is propagated to the thread executing the task.
- **ContextService:** Used to create dynamic proxy objects that capture the context of a container and enable applications to run within that context at a later time or be submitted to a Managed Executor Service. The context of the container is propagated to the thread executing the task.

For more detailed information, see Concurrency Utilities for Java EE in the *The Java EE 8 Tutorial*. Also, see JSR 236: Concurrency Utilities for Java EE.

- Concurrency 1.0 Code Examples in WebLogic Server

Concurrency 1.0 Code Examples in WebLogic Server

When you install WebLogic Server Code Examples, the examples source code is placed in the *EXAMPLES_HOME*\examples\src\examples directory. The default path of *EXAMPLES_HOME* is *ORACLE_HOME*\wlserver\samples\server. From this directory, you can access the source code and instruction files for the Concurrency 1.0 examples.

The *ORACLE_HOME*\user_projects\domains\wl_server directory contains the WebLogic Server code examples domain. See Sample Applications and Code Examples in *Understanding Oracle WebLogic Server*.

- **Using Concurrency ContextService**: Demonstrates how to use the `ContextService` interface to create dynamic proxy objects.

  EXAMPLES_HOME/examples/src/examples/javaee7/concurrency/dynamicproxy

- **Using Concurrency Executor**: Demonstrates how to use `javax.enterprise.concurrent.ManagedExecutorService` for submitting tasks.

  EXAMPLES_HOME/examples/src/examples/javaee7/concurrency/executor

- **Using Concurrency Schedule**: Demonstrates how to use `javax.enterprise.concurrent.ManagedScheduledExecutorService` for submitting delayed or periodic tasks.

  EXAMPLES_HOME/examples/src/examples/javaee7/concurrency/schedule

- **Using Concurrency Threads**: Demonstrates how to use `javax.enterprise.concurrent.ManagedThreadFactory` to obtain a thread from the Jakarta EE container.

  EXAMPLES_HOME/examples/src/examples/javaee7/concurrency/threads

Oracle recommends that you run these examples before programming your own applications that use concurrency.

How Concurrent Managed Objects Provide Concurrency for WebLogic Server Containers

Learn how WebLogic Server provides concurrency capabilities to Jakarta EE applications by associating the Concurrency Utilities API with the Work Manager to make threads container-managed.

How WebLogic Server Handles Asynchronous Tasks in Application Components

With JSR 236 Concurrent Utilities, WebLogic Server can recognize the asynchronous tasks in a server application component, and then manages them by:

- Providing the proper execution context. See CMO Context Propagation.
- Submitting tasks to the single server-wide self-tuning thread pool to make them prioritized based on defined rules and runtime metrics. See Self Tuning for CMO Tasks.
- Interrupting the thread that the task is executed in, when the component that created the task is shutting down. See Threads Interruption When CMOs Are Shutting Down.
- Limiting the number of new running threads to be created by managed objects when the task is not suitable to be dispatched to the self-tuning thread pool. See CMO Constraints for Long-Running Threads.

Concurrent Managed Objects (CMOs)

In WebLogic Server, asynchronous task management is provided by four types of Concurrent Managed Objects (or CMOs).

Table 5-1 summarizes the CMOs that provide asynchronous task management.

Table 5-1 CMOs that Provide Asynchronous Task Management

Managed Object	Context Propagation	Self Tuning	Thread Interruption While Shutting Down	Limit of Concurrent Long-Running New Threads
Managed Executor Service (MES)	Contexts are propagated based on configuration. See CMO Context Propagation.	Only short-running tasks are dispatched to the single self-tuning thread pool by a specified Work Manager. See Self Tuning for CMO Tasks.	When Work Manager is shutting down, all the unfinished tasks will be canceled. See Threads Interruption When CMOs Are Shutting Down.	The maximum number of long-running threads created by MES/MSES can be configured to avoid excessive number of these threads making negative effect on server. See CMO Constraints for Long-Running Threads.
Managed Scheduled Executor Service (MSES)	Contexts are propagated based on configuration. See CMO Context Propagation.	Same behavior as MES. See Self Tuning for CMO Tasks.	Same behavior as MES. See Threads Interruption When CMOs Are Shutting Down.	Same behavior as MES. See CMO Constraints for Long-Running Threads.
Context Service	Contexts are propagated based on configuration. See CMO Context Propagation.	n/a	n/a	n/a
Managed Thread Factory (MTF)	Contexts are propagated based on configuration. See CMO Context Propagation.	Threads returned by the `newThread()` method are not from the single self-tuning thread pool and will not be put into the thread pool when the task is finished. See Self Tuning for CMO Tasks.	Threads created by the `newThread()` method will be interrupted when the MTF is shutting down. See Threads Interruption When CMOs Are Shutting Down.	The maximum number of new threads created by MTF can be configured to avoid excessive number of these threads making negative effect on server. See CMO Constraints for Long-Running Threads.

The following are the three types of JSR 236 CMOs in WebLogic Server, each one characterized by its scope, and how it is defined and used:

- Default Jakarta Concurrency Objects – Required by the Jakarta EE standard that default resources be made available to applications, and defines specific JNDI names for these default resources.
- Customized CMOs in Configuration Files – Can be defined at the application and module level or referenced from an application component environment (ENC) that is bound to JNDI.
- Global CMO Templates – Can be defined globally as templates in the domain's configuration by using the WebLogic Remote Console and configuration MBeans.

Similar to Work Managers, global CMO templates can be defined at the domain or server level using the WebLogic Remote Console or configuration MBeans. See Using MBeans to Configure CMO Templates.

Also, see Concurrent Managed Object Templates in the *Oracle WebLogic Remote Console Online Help*.

CMOs versus CommonJ API

The CommonJ API (`commonj.work`) in WebLogic Server provides a set of interfaces that allow an application to execute multiple work items concurrently within a container. CMOs and CommonJ APIs operate at the same level: they both dispatch tasks to Work Managers and programmatically handle work from within an application. However, there are distinct differences between CMOs and the CommonJ API, such as:

- CommonJ API is WebLogic specific and CMOs have been standardized.
- CommonJ API provides functions similar to the CMO Managed Executor Service and Managed Scheduled Executor Service, but it does not provide CMO functions like the Managed Thread Factory and the Context Service.

For information about using the CommonJ API, see Using the Timer and Work Manager API in *Developing CommonJ Applications for Oracle WebLogic Server*.

CMO Context Propagation

This section explains the four context types that are propagated for CMOs and the context invocation points in WebLogic Server for MES and MSES managed objects.

- Propagated Context Types
- Contextual Invocation Points

Propagated Context Types

Table 5-2 summarizes the context types that are propagated for the four types of managed objects.

Table 5-2 Propagated Context Types

Context Type	Description	Context Tasks Run with...
JNDI	JNDI namespace	For MES, MSES, and ContextService, tasks can access the application-scoped JNDI tree (such as `java:app`, `java:module`, `java:comp`) of the submitting thread. For MTF, tasks can access application-scoped JNDI tree of the component that created the ManagedThreadFactory instance.
ClassLoader	Context Class loader	For MES, MSES, and ContextService, tasks run with the context classloader of the submitting thread. For MTF, tasks run with the classloader of the component that created the ManagedThreadFactory instance

Table 5-2 (Cont.) Propagated Context Types

Context Type	Description	Context Tasks Run with...
Security	Subject identity	For MES, MSES, and ContextService, tasks run with the subject identity of the submitting thread.For MTF, tasks run with the anonymous subject.
WorkArea	WorkArea contexts with PropagationMode `WORK`	For MES, MSES, and Context Service there is a new WorkArea context type, and so all tasks run with a WorkContextMap, which contains all the submitting thread's contexts with WORK mode. For MTF, all tasks run with an empty WorkContextMap. **Note:** While the WorkContextMap is a new instance, the contained values are not, so updating the contents of the values can affect the contents of the submitting thread.

Contextual Invocation Points

Table 5-3 summarizes the callback methods of the Contextual Invocation Points in WebLogic Server and the context that the Contextual Invocation Point runs with, for the MES and MSES managed objects.

Table 5-3 Contextual Invocation Points

Concurrent Managed Objects	Contextual Invocation Points	Context with which the Contextual Invocation Point Runs
ManagedExecutorService	callback method: `javax.enterprise.concurrent.ManagedTaskListener`	The Contextual Invocation Points run with the context of the application component instance that called the `submit()`, `invokeAll()`, `invokeAny()` methods.
ManagedScheduledExecutorService	callback methods: `javax.enterprise.concurrent.ManagedTaskListener` and `javax.enterprise.concurrent.Trigger`	The application component instance that called the `submit()`, `invokeAll()`, `invokeAny()`, `schedule()`, `scheduleAtFixedRate()`, `scheduleWithFixedDelay()` methods.

Self Tuning for CMO Tasks

Short-running tasks submitted to the MES or the MSES are dispatched to the single self-tuning thread pool by associating with the Work Manager specified in deployment descriptors.

The execution of the tasks will be consistent with the rules defined for the specified Work Manager. For tasks submitted to the execute method in MES and MSES, if the Work Manager's overload policy rejects the task, the following events will occur:

- The `java.util.concurrent.RejectedExecutionException` will be thrown in the `submit` or `execute` method.
- The overload reason parameter passed to `weblogic.work.Work` will be set to the `RejectedExecutionException`.
- If the user registered the task with the `ManagedTaskListener`, the listener will not be notified because the user can receive the overload message through the `RejectedExecutionException`.

  Note: A `ManagedTaskListener` is used to monitor the state of a task's future. For more information see, `Package javax.enterprise.concurrent`.

For the `invokeAll()` and `invokeAny()` methods in the MES and MSES, and for any of the submitted tasks that are rejected by the Work Manager overload policy, the following events will occur:

- The user-registered `ManagedTaskListener`'s `taskSubmitted()` method will be called.
- The user-registered `ManagedTaskListener`'s `taskDone()` method will be called and the `throwableParam` will be `javax.enterprise.concurrent.AbortedException`.
- The overload reason parameter passed to `weblogic.work.Work` will be set to the `AbortedException`.

For the `schedule()`, `scheduleAtFixRate()`, `scheduleAtFixDelay()`, and `schedule(Trigger)()` methods, if the task is rejected by the Work Manager's overload policy, the following events will occur:

- The user-registered `ManagedTaskListener`'s `taskDone()` method will be called, the `throwableParam` will be `javax.enterprise.concurrent.AbortedException`.
- The overload reason parameter passed to `weblogic.work.Work` will be set to the `AbortedException`.
- If the task is periodic, the next run of task will still be scheduled.

Threads Interruption When CMOs Are Shutting Down

When either the MES or MSES is shut down:

- None of the waiting tasks will be executed.
- All the running threads will be interrupted. The user should check the `Thread.isInterrupted()` method and terminate their tasks because WebLogic Server will not force it to terminate.
- An executor returned Future object will throw the `java.util.concurrent.CancellationException()` if the `Future.get()` method is called.
- User registered `ManagedTaskListener`'s `taskAborted()` method will be called and `paramThrowable` will be the `CancellationException()`.

When the MTF is shut down:

- All threads that are created using the `newThread()` method are interrupted. Calls to the `isShutdown()` method in the `ManageableThread` interface on these threads return true.

- All subsequent calls to the `newThread()` method throw a `java.lang.IllegalStateException`.

For the ContextService, no thread is interrupted. However, all invocations to any of the proxied interface methods will fail with a `java.lang.IllegalStateException`.

CMO Constraints for Long-Running Threads

Long-running tasks submitted to MES and MSES, and the calling of `newThread()` method of MTF need to create new threads that will not be managed as a part of the self-tuning thread pool. An excessive number of running threads can have a negative affect on the server performance and stability. Therefore, configurations are provided to specify the maximum number of running threads that are created by the concurrency utilities API.

- Setting Limits for Maximum Concurrent Long Running Requests
- Setting Limits for Maximum Concurrent New Threads

Setting Limits for Maximum Concurrent Long Running Requests

The limit of concurrent long-running requests submitted to MES and MSES can be specified in managed object and server levels. All levels of configurations are independent and the maximum of concurrent long-running requests cannot exceed any of them.

Table 5-4 summarizes the limit of concurrent long-running requests with the `<max-concurrent-long-running-requests>` element that can be defined in the deployment descriptors.

Table 5-4 Limit of Concurrent Long-running Requests

Scope	Deployment Descriptor	Description	`<max-concurrent-long-running-requests>` Element Details
Server	In `config.xml`: As the sub-element of `<domain><server>` or `<domain><server-template>`	Limit of concurrent long-running requests specified for that server.	Optional Range: [0-65534]. When out of range, the default value will be used. Default value: `100`
Managed Object	In `weblogic-application.xml`, `weblogic-ejb-jar.xml`, or `weblogic.xml`: As the sub-element of `<managed-executor-service>` or `<managed-scheduled-executor-service>` In `config.xml`: As the sub-element of `<managed-executor-service-template>` or `<managed-scheduled-executor-service-template>`	Limit of concurrent long-running requests specified for that MES or MSES.	Optional Range: [0-65534]. When out of range, the default value will be used. Default value: `10`

When the specified limit exceeds, MES or MSES takes the following actions for the new long-running tasks submitted to them:

- The `java.util.concurrent.RejectedExecutionException` will be thrown when calling the task submission API.
- If the user registered the task with the `ManagedTaskListener`, then this listener will not be notified because the `submit` method fails.

Note that the above rule is not applied for the `invokeAll()` and `invokeAny()` methods. If any of the tasks submitted by these methods is rejected by the specified limit, the following events will occur:

- The user-registered `ManagedTaskListener`'s `taskSubmitted()` method will be called.
- The user-registered `ManagedTaskListener`'s `taskDone()` method will be called and the `throwableParam` will be `javax.enterprise.concurrent.AbortedException`.
- Other submitted tasks will not be affected.
- The method will *not* throw the `RejectedExecutionException`.

Example 5-1 demonstrates how the value specified for the `<max-concurrent-long-running-requests>` element in the `config.xml` can affect the maximum number of long-running requests.

Example 5-1 Sample Placements of max-concurrent-long-running-requests in config.xml

```
<domain>
    <server>
        <name>myserver</server>
        <max-concurrent-long-running-requests>50</max-concurrent-long-running-requests>
(place 1)
    </server>
    <max-concurrent-long-running-requests>10</max-concurrent-long-running-requests>
(place 2)
    <server-template>
        <name>mytemplate</name>
        <max-concurrent-long-running-requests>50</max-concurrent-long-running-requests>
(place 3)
    </server-template>
</domain>
```

- `place 1` – Affects the MES and MSES defined in the server instance `myserver`. All the MES and MSES running in that server instance can only create a maximum of 50 long-running requests in total.
- `place 2` – Only affects MES and MSES defined in the domain. All the MES and MSES running in the domain can create a maximum of 10 long-running requests in total.
- `place 3` – Affects MES & MSES defined in the server instances that apply to the template `mytemplate`. All the MES and MSES running in that server instance can only create a maximum of 50 long-running requests in total.

Example 5-2 demonstrates a sample configuration of `max-concurrent-long-running-requests`.

Example 5-2 Sample Configurations of max-concurrent-long-running-requests

```
server1(100)
    |---application1
          |---managed-scheduled-executor-service1(not specified)
```

```
        |---module1
              |---managed-executor-service1(20)
              |---managed-scheduled-executor-service2(not specified)
|---application2
```

In the following cases, none of the limits are exceeded and the above actions will not be taken:

- Assume that 120 long-running tasks are submitted to `managed-executor-service1`, out of which 115 are finished and 5 are being executed. If one more long-running task is submitted to `managed-executor-service1`, it will be executed as no limit is exceeded.

In the following cases, one of the limits is exceeded and the above actions will be taken:

- Assume that 10 long-running tasks are being executed by `managed-scheduled-executor-service1`. If one more long-running task is submitted to `managed-scheduled-executor-service1`, then the limit of `managed-scheduled-executor-service1` is exceeded.
- Assume that 10 long-running tasks are being executed by `application1` and 90 are being executed by `application2`. If one more long-running task is submitted to `application1` or `application2`, then the limit of `server1` is exceeded.

Setting Limits for Maximum Concurrent New Threads

The limit of concurrent new running threads created by calling the `newThread()` method of the MTF can be specified in a managed object, domain, and server level. All levels of configurations are independent and the maximum of the concurrent new running threads cannot exceed any of them.

A running thread is a thread that is created by the MTF and has not finished its `run()` method.

Table 5-5 summarizes the limit of concurrent new running threads with an element `<max-concurrent-new-threads>` that can be defined in the deployment descriptors.

Table 5-5 Limit of Concurrent New Running Threads

Scope	Deployment Descriptor	Description	<max-concurrent-new-threads> Element Details
Server	In `config.xml`: As the sub-element of `<domain><server>` or `<domain><server-template>`	Limit of concurrent new running threads specified for that server.	Optional Range: [0-65534]. When out of range, the default value will be used Default value: `10`

Table 5-5 (Cont.) Limit of Concurrent New Running Threads

Scope	Deployment Descriptor	Description	<max-concurrent-new-threads> Element Details
Managed Object	In `weblogic-application.xml`, `weblogic-ejb-jar.xml`, or `weblogic.xml`: As the sub-element of `<managed-executor-service>` or `<managed-scheduled-executor-service>` In `config.xml`: As the sub-element of `<managed-executor-service-template>` or `<managed-scheduled-executor-service-template>`	Limit of concurrent new running threads specified for that `ManagedThreadFactory`.	Optional Range: [0-65534]. When out of range, the default value will be used Default value: `10`

When the specified limit is exceeded, calls to the `newThread()` method of the MTF will return `null` to be consistent with the `ThreadFactory.newThread` Javadoc.

For a sample snippet of using `max-concurrent-new-threads`, see Deployment Descriptor Examples.

Default Jakarta Concurrency Objects

The Jakarta standard specifies that certain default resources be made available to applications, and defines specific JNDI names for these default resources. WebLogic Server makes these names available through the use of logical JNDI names, which map Jakarta standard JNDI names to specific WebLogic Server resources.

- Default Managed Executor Service
- Default Managed Scheduled Executor Service
- Default Context Service
- Default Managed Thread Factory

Default Managed Executor Service

There is a default MES instance for each application. It is automatically bound to the default JNDI name of `java:comp/DefaultManagedExecutorService` of all the sub-components when deployed.

The default MES:

- Uses the default Work Manager as the dispatch policy.
- Propagates all the context-info.
- The long-running request limit default is `10`.
- The long-running thread priority defaults to `normal`.

You can also use the default MES in applications with the `@Resource` annotation. For example:

```
package com.example;
   public class TestServlet extends HttpServlet {
   @Resource
   private ManagedExecutorService service;
```

Overriding the Default MES

The behavior of the default MES can be overridden by:

- Defining an executor template named `DefaultManagedExecutorService` in the `config.xml`. All applications will use this template to create a default MES.
- Defining a custom `managed-executor-service` in the `weblogic-application.xml`, using either deployment descriptors or annotations. This will also override the default MES definition in the `config.xml` in the application. See Custom Managed Executor Service Configuration Elements.

You cannot define a default executor named `DefaultManagedExecutorService` in the `weblogic.xml` or `weblogic-ejb-jar.xml`. Doing so will cause the deployment to fail.

Default Managed Scheduled Executor Service

The default MSES instance is similar to the default MES instance, but is automatically bound to the default JNDI name of `java:comp/DefaultManagedScheduledExecutorService` of all the sub-components when deployed. It has the same default settings and propagates all the context information.

You can also use the default MSES in applications with the `@Resource` annotation. For example:

```
package com.example;
  public class TestServlet extends HttpServlet {
  @Resource
  private ManagedScheduledExecutorService service;
```

Overriding the Default MSES

The behavior of the default MSES can be overridden by:

- Defining a scheduled executor template named `DefaultManagedScheduledExecutorService` in the `config.xml`. All applications will use this template to create a default MSES.
- Defining a custom `<managed-scheduled-executor-service>` in the `weblogic-application.xml`, using either deployment descriptors or annotations. This will also override the default MSES definition in the `config.xml` in the application. See Custom Managed Scheduled Executor Service Configuration Elements.

You cannot define a default scheduled executor named `DefaultManagedExecutorService` in the `weblogic.xml` or `weblogic-ejb-jar.xml`. Doing so will cause the deployment to fail.

Default Context Service

There is a default context service instance for each application. It is automatically bound to the default JNDI name of `java:comp/DefaultContextService` of all the sub-components when deployed, and propagates all types of supported contexts.

The default Context Service can also be bound to `java:comp/env/concurrent/cs` under an application component environment (ENC) using the `resource-env-ref` or `@Resource` annotation.

Note that the behavior of the default context service cannot be overridden.

Example 5-3 shows how to use the default context service in a `webl.xml` file using the `resource-env-ref` element.

Example 5-3 Using the Default Context Service with <resource-env-ref> in a Web App

```
<!-- web.xml -->
<resource-env-ref>
  <resource-env-ref-name>concurrent/cs</resource-env-ref-name>
  <resource-env-ref-type>javax.enterprise.concurrent.ContextService</resource-env-ref-
type>
</resource-env-ref>
```

Example 5-4 shows how to use the default context service in a servlet with the `@Resource` annotation.

Example 5-4 Using the Default Context Service with @Resource in a Servlet

```
// when using @Resource, the following 2 ways are correct.
@Resource(lookup="java:comp/env/concurrent/cs")
// @Resource(name="concurrent/cs")
private ContextService service;

// when using JNDI Naming Context to lookup:
// initialContext.lookup("java:comp/env/concurrent/cs")
```

Default Managed Thread Factory

There is a default MTF instance for each application. It is automatically bound to the default JNDI name of `java:comp/DefaultManagedThreadFactory` of all the sub-components when deployed.

The default MTF:

- Propagates all types of supported contexts for new threads.
- The default priority for long-running threads created by `newThread()` is `normal`.
- The default limit for running concurrent new threads is `10`.

You can also use the default MTF in applications with the `@Resource` annotation. For example:

```
package com.example;
  public class TestServlet extends HttpServlet {
  @Resource
  private ManagedThreadFactory service;
```

Overriding the Default MTF

The behavior of the default MTF can be overridden by:

- Defining a thread factory template named `DefaultManagedThreadFactory` in the `config.xml`. All applications will use this template to create a default MTF.
- Defining a custom `managed-thread-factory` in the `weblogic-application.xml`, using either deployment descriptors or annotations. This will also override the default MTF

definition in the `config.xml` in the application. See Custom Managed Thread Factory Configuration Elements.

You cannot define a default thread factory named `DefaultManagedThreadFactory` in the `weblogic.xml` or `weblogic-ejb-jar.xml`. Doing so will cause the deployment to fail.

Customized CMOs in Configuration Files

You can define the customized CMOs at the application and module level, or referenced from an application component environment (ENC) that is bound to JNDI.

> **Note:**
>
> In the current release, a custom Context Service cannot be configured.

- Defining CMOs in WebLogic Configuration Files
- Binding CMOs to JNDI Under an Application Component Environment
- Custom Managed Executor Service Configuration Elements
- Custom Managed Scheduled Executor Service Configuration Elements
- Custom Managed Thread Factory Configuration Elements
- Transaction Management for CMOs

Defining CMOs in WebLogic Configuration Files

Customized CMOs can be defined at the application and module level in one of these configuration files:

- `weblogic-application.xml`—CMOs specified at the application level can be assigned to that application, or any component of that application.
- `weblogic-ejb-jar.xml` or `weblogic.xml`—CMOs specified at the component level can be assigned to that component.

Binding CMOs to JNDI Under an Application Component Environment

Executor and thread factory CMOs can also be bound to JNDI under an application component environment (ENC) using the `resource-env-ref` element or the `@Resource` annotation. The `resource-env-ref` referencing a CMO can only be defined in the `web.xml`, `ejb-jar.xml`, or `application.xml`.

The four ENC namespaces (`java:comp`, `java:module`, `java:application`, and `java:global`) are supported for `resource-env-ref-name` and `@Resource`.

If you bind an executor in an application, *AppA*, to the `java:global` JNDI namespace, the executor can be looked up and used by another application, *AppB*. Tasks submitted by *AppB* are canceled when *AppA* or *AppB* is shutdown.

- JNDI Binding Using <resource-env-ref>
- JNDI Binding Using @Resource
- Updated Schemas for Custom CMO Modules

- Updated System Module Beans for CMOs

JNDI Binding Using <resource-env-ref>

Example 5-5 demonstrates how to map an MES named `MyExecutor` to the `java:comp/env` JNDI namespace.

Example 5-5 Binding an Executor to JNDI Using <resource-env-ref>

```
weblogic.xml
   <resource-env-description>
      <resource-env-ref-name>concurrent/MyExecutor</resource-env-ref-name>
      <resource-link>MyExecutor</resource-link>
   </resource-env-description>

   web.xml
   <resource-env-ref>
      <resource-env-ref-name>concurrent/MyExecutor</resource-env-ref-name>
      <resource-env-ref-type>javax.enterprise.concurrent.ManagedExecutorService</
resource-env-ref-type>
   </resource-env-ref>
```

In the `weblogic.xml`, the `resource-link` element specifies which executor is being mapped, which in Example 5-5 is named `MyExecutor`.

Executors defined in the `weblogic.xml` are searched first, followed by the `weblogic-application.xml`, and then the `managed-executor-service-template` in the `config.xml` to find a executor name attribute that matches the one specified in `resource-link`.

If the `resource-env-description` is defined in the `weblogic-ejb-jar.xml`, then the `weblogic-ejb-jar.xml` is searched first, then the `weblogic-application.xml`, and then the `config.xml`.

JNDI Binding Using @Resource

The mapping rules for the `@Resource` annotation are equivalent to those for `resource-env-ref`, but uses different naming conventions:

- `resource-env-ref-name` is the `name` attribute value in `@Resource`.
- `resource-link` is equivalent to the `mappedName` attribute value defined in `@Resource`.

If `@Resource` is used under a web component, it is equivalent to define a `resource-env-ref` under `web.xml`.

If `@Resource` is used under an EJB component, it is equivalent to define a `resource-env-ref` under `ejb-jar.xml`.

The annotation can also be used on class or methods as defined in the Jakarta specification.

Example 5-5 using the `resource-env-ref` definition is equivalent to Example 5-6 using `@Resource`.

Example 5-6 Binding an Executor to JNDI Using @Resource

```
   package com.example;
   public class TestServlet extends HttpServlet {
   @Resource(name="concurrent/MyExecutor" mappedName="MyExecutor")
   private ManagedExecutorService  service;
```

In this example, if the `mappedName` attribute of `@Resource` is not specified, then the default executor is used.

If you define both the `resource-env-ref` and `@Resource`, and if `resource-env-ref-name` and `name` attribute of `@Resource` are the same, then the `resource-env-ref` defined executor will be injected into the `@Resource` field.

You can also use `@Resource` with a lookup attribute or `InitialContext.lookup` to find a executor bound by `resource-env-ref`.

Updated Schemas for Custom CMO Modules

The following WebLogic Server schemas include elements for configuring the CMO deployment descriptors:

- `weblogic-javee.xsd` – Describes common elements shared among all WebLogic-specific deployment descriptors:

  http://xmlns.oracle.com/weblogic/weblogic-javaee/1.8/weblogic-javaee.xsd

- `weblogic-application.xsd` – The WebLogic Server-specific deployment descriptor extension for the `application.xml` Jakarta EE deployment descriptor, where you configure features such as shared Jakarta EE libraries referenced in an application and EJB caching.

  See weblogic-application.xml Deployment Descriptor Elements in *Developing Applications for Oracle WebLogic Server*.

- `weblogic-web-app.xsd` – The WebLogic Server-specific deployment descriptor for Web applications.

  See weblogic.xml Deployment Descriptor Elements in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

- `weblogic-ejb-jar.xsd` – The WebLogic-specific XML Schema-based (XSD) deployment descriptor file for EJB deployments.

  See weblogic-ejb-jar.xml Deployment Descriptor Reference in *Developing Enterprise JavaBeans, Version 3.2, for Oracle WebLogic Server*.

Example 5-7 shows the CMO-related elements in the `weblogic-web-app.xsd`.

Example 5-7 CMO Elements in weblogic-web-app.xsd

```
<xs:complexType name="weblogic-web-appType">
    <xs:sequence>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      ...
      <!-- added for JSR236 -->
      <xs:element name="managed-executor-service" type="wls:managed-executor-
serviceType" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="managed-scheduled-executor-service" type="wls:managed-scheduled-
executor-serviceType" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="managed-thread-factory" type="wls:managed-thread-factoryType"
minOccurs="0" maxOccurs="unbounded"/>
      <!-- added end -->
      ...
```

Updated System Module Beans for CMOs

The following WebLogic Server system module beans include attributes for configuring CMOs in applications and modules:

- `ManagedExecutorServiceBean`
- `ManagedScheduledExecutorServiceBean`

- `ManagedThreadFactoryBean`
- `WeblogicApplicationBean`

See the WebLogic Server System Module MBeans section in the *MBean Reference for Oracle WebLogic Server*.

Custom Managed Executor Service Configuration Elements

This section defines the configuration elements for a Managed Executor Service.

Table 5-6 Managed Executor Service Configuration Elements

Name	Description	Required	Default Value	Range
`name`	The name of the MES. An MES with the same name cannot be configured in the same scope. For example, if the same MES name is used in an application or module scope, the deployment of the application will fail. An MES can have the same name as the other types of managed objects, such as a `ContextService`, in any scope, and there will be no relationship between them. An MES with the same name can only be configured in different scopes: • If there is more than one MES with the same name configured in the server template or application scope, the application scope MES will override the server template. • If there is an MES A defined in the module scope which has the same name as the executor B defined in the server template or application scope. A and B will both exist. The executor which is used is determined by the location referencing the executor.	Yes	n/a	An arbitrary non-empty string.
`dispatch-policy`	The name of the Work Manager. The rule of which Work Manager should be used is: • Search module scope Work Manager first, if the `ManagedExecutorService` is defined in the module scope. • If not found, search application level. • If still not found, the default Work Manger is used. (This behavior is consistent with the servlet and EJB's dispatch policy resolving strategy.)	No	Default Work Manager	n/a
`max-concurrent-long-running-requests`	Maximum number of concurrent long-running tasks. See Setting Limits for Maximum Concurrent Long Running Requests.	No	10	[0-65534]. When out of range, the default value will be used.

Table 5-6 (Cont.) Managed Executor Service Configuration Elements

Name	Description	Required	Default Value	Range
`long-running-priority`	An integer that specifies the long-running daemon thread's priority. If specified, all long-running threads will be affected. See Setting Limits for Maximum Concurrent New Threads.	No	`Thread.NORM_PRIORITY`	1-10 Range between `Thread.MIN_PRIORITY` and `Thread.MAX_PRIORITY`. When out of range, the default value will be used.

- Deployment Descriptor Examples

Deployment Descriptor Examples

Example 5-8 is an example of a custom MES definition in a Web application's `weblogic.xml` file.

Example 5-8 Using Deployment Descriptor to Define a Custom MES in an Application

```
<!-- weblogic.xml -->
   <managed-executor-service>
       <name>MyExecutor</name>
       <dispatch-policy>MyWorkManager</dispatch-policy>
       <long-running-priority>10</long-running-priority>
       <max-concurrent-long-running-requests>10</max-concurrent-long-running-requests>
   </managed-executor-service>
```

Example 5-9 is an example of a custom MES reference in the `weblogic.xml` descriptor using the `<resource-env-ref>` element.

Example 5-9 Referencing a Custom MES Using <resource-env-ref> in an Application

```
weblogic.xml
  <resource-env-description>
    <resource-env-ref-name>concurrent/MyExecutor</resource-env-ref-name
    <resource-link>MyExecutor</resource-link>
  </resource-env-description>
```

Example 5-10 is an example of a custom MES reference in a `webl.xml` file using the `<resource-env-ref>` element.

Example 5-10 Referencing a Custom MES Using <resource-env-ref> in a Web App

```
web.xml
  <resource-env-ref>
    <resource-env-ref-name>concurrent/MyExecutor</resource-env-ref-name
    <resource-env-ref-type>javax.enterprise.concurrent.ManagedExecutorService</resource-
env-ref-type>
  </resource-env-ref>
```

Example 5-11 is an example of a custom MES reference in a servlet using the `@Resource` annotation.

Example 5-11 Referencing a Custom MES in a Servlet Using @Resource in a Servlet

```
package com.example;
public class TestServlet extends HttpServlet {
```

```
@Resource(name="concurrent/MyExecutor" mappedName="MyExecutor")
private ManagedExecutorService  service;
```

Custom Managed Scheduled Executor Service Configuration Elements

This section defines the configuration elements for a Managed Scheduled Executor Service.

Table 5-7 Managed Scheduled Executor Service Configuration Elements

Name	Description	Required	Default Value	Range
`name`	The name of the MSES. For naming convention rules, see Table 5-6.	Yes	n/a	An arbitrary non-empty string.
`dispatch-policy`	The name of the Work Manager. For Work Manager usage rules, see Table 5-6.	No	Default Work Manager	n/a
`max-concurrent-long-running-requests`	Maximum number of concurrent long-running tasks. See Setting Limits for Maximum Concurrent Long Running Requests.	No	`10`	[0-65534]. When out of range, the default value is used.
`long-running-priority`	An integer that specifies the long-running daemon thread's priority. If specified, all long-running threads will be affected. See Setting Limits for Maximum Concurrent New Threads.	No	`5` `Thread.NORM_PRIORITY`	1-1 Range between `Thread.MIN_PRIORITY` and `Thread.MAX_PRIORITY`. When out of range, the default value is used.

- ScheduledFuture.get() Method
- Deployment Descriptor Examples

ScheduledFuture.get() Method

The `ScheduledFuture.get()` method will block until the latest run of the task finishes. For example, if the `Trigger` method requires the task being scheduled to run two times (that is, `Trigger.getNextRunTime` returns null on the third call), and the first run of the task is finished at time *A*, the second run of the task is finished at time *B*, then:

- If `Future.get()` is called before time *A*, it will wait for the first run to finish and return the first run result. If it is called after time *A* and before time *B*, it will wait to the second run finish and return the second run's result.
- If `Future.get()` is called after time *B*, it will immediately return the second run result. Also, if the first run fails and throws a exception, then the first `Future.get()` call will throw that

exception and the second run will still be scheduled (this is different with `scheduleAtFixRate`). If the `Trigger.skipRun` returns `true` on the first run, then the first `Future.get()` call will throw a `SkipException`.

Deployment Descriptor Examples

Example 5-12 is an example of a custom MSES definition in a Web application's `weblogic.xml` file.

Example 5-12 Using Deployment Descriptor to Define a Custom MSES in an Application

```
<!-- weblogic.xml -->
   <managed-scheduled-executor-service>
       <name>MyScheduledExecutor</name>
       <dispatch-policy>MyExecutor</dispatch-policy>
   </managed-scheduled-executor-service>
```

Custom Managed Thread Factory Configuration Elements

This section defines the configuration elements for a Managed Thread Factory.

Table 5-8 Managed Thread Factory Configuration Elements

Name	Description	Required	Default Value	Range
`name`	The name of the MTF. For naming convention rules, see Table 5-6.	Yes	n/a	An arbitrary non-empty string.
`priority`	The priority to assign to the thread. The higher the number, the higher the priority. See Setting Limits for Maximum Concurrent New Threads.	No	`5` `Thread.NORM_PRIORITY`	1-10 Range between `Thread.MIN_PRIORITY` and `Thread.MAX_PRIORITY`. When out of range, the default value is used.
`max-concurrent-new-threads`	The maximum number of threads created by the MTF and are still executing the `run()` method of the tasks. See Setting Limits for Maximum Concurrent New Threads.	No	10	[0-65534] When out of range, the default value is used.

Contexts of Threads Created by MTF

According to JSR 236, the Managed Thread Factory is different from the other managed objects because when the thread is started using the `Thread.start()` method, the runnable that is executed will run with the context of the application component instance that created the `ManagedThreadFactory` instance. Therefore, the context of the runnable depends on the application component that created the MTF instance.

In WebLogic Server, new MTF instances are created when an application or a component (that is, a web module or an EJB) is started, as follows:

1. A default MTF is created by that component.

2. If there is a `@Resource` annotation to get an MTF, an MTF instance is created by that component.
3. If there is a `<resource-env-ref>` defined in the `web.xml` or `ejb-jar.xml`, and there is also a corresponding `<resource-env-description>` defined in the `weblogic.xml` or `weblogic-ejb-jar.xml` with a `<resource-link>` for an MTF, then an MTF instance is created by that component.
4. If there is a `<resource-env-ref>` defined in the `application.xml`, and there is also a corresponding `<resource-env-description>` defined in the `weblogic-application.xml` with a `<resource-link>` for an MTF, then an MTF instance is created by that application.

When an MTF is created by a component in the case of items 1, 2, and 3 listed above, the runnable runs with the context of that component, as follows:

- **Classloader:** The classloader of that component.
- **JNDI:** The JNDI tree of that component that contains `java:app`, `java:module`, and `java:comp`.
- **Security:** Fixed to be the anonymous subject because there is no component-specific subject.
- **WorkArea:** Fixed to be an empty `WorkContextMap` because there is no component-specific `WorkContextMap`.

When an MTF is created by an application in the case of item number 4, listed above, the runnable runs with the context of that application as follows:

- **Classloader:** The classloader of that application.
- **JNDI:** The JNDI tree of that component that contains `java:app`, but without `java:module` and `java:comp`.
- **Security:** Fixed to be the anonymous subject because there is no application-specific subject.
- **WorkArea:** Fixed to be an empty `WorkContextMap` because there is no application-specific `WorkContextMap`.

Deployment Descriptor Examples

Example 5-13 is an example of a custom MTF definition in a Web application's `weblogic.xml` file.

Example 5-13 Using Deployment Descriptors to Define a Custom MTF in an Application

```
<!-- weblogic.xml -->
   <managed-thread-factory>
       <name>factory1</name>
       <priority>3</priority>
       <max-concurrent-new-threads>20</max-concurrent-new-threads>
   </managed-executor-service>
```

Example 5-9 is an example of a custom MTF reference in a Web application's `weblogic.xml` file.

Example 5-14 Referencing a Custom MTF Using <resource-env-ref> in an Application

```
weblogic.xml
  <resource-env-description>
    <resource-env-ref-name>ref-factory1</resource-env-ref-name
```

```
    <resource-link>factory1</resource-link>
  </resource-env-description>
```

Example 5-10 is an example of a custom MTF reference in a Web application's `weblogic.xml` file.

Example 5-15 Referencing a Custom MTF Using <resource-env-ref> in a Web App

```
web.xml
  <resource-env-ref>
    <resource-env-ref-name>ref-factory1</resource-env-ref-name
    <resource-env-ref-type>javax.enterprise.concurrent.ManagedThreadFactory</resource-
env-ref-type>
  </resource-env-ref>
```

Example 5-11 is an example of a custom MTF reference in a servlet using the `@Resource` annotation.

Example 5-16 Referencing a Custom MTF Using @Resource in a Servlet

```
package com.example;
public class TestServlet extends HttpServlet {
@Resource(lookup="java:comp/env/ref-factory1")
private ManagedThreadFactory factory;
```

Transaction Management for CMOs

This section explains how transactions are managed by the WebLogic Server for CMOs.

- Transaction Management for MES and MSES
- Transaction Management for Context Service
- Transaction Management for MTF

Transaction Management for MES and MSES

When using an MES, transactions are managed as follows:

- There are no transactions running in the Work Manager thread before the task is begun.
- The `UserTransaction.getStatus()` method is always `Status.STATUS_NO_TRANSACTION` unless the Transaction API is used to start a new transaction.
- User should always finish its transaction in user tasks; otherwise, the transaction will be rolled back.

Therefore `ManagedTask.TRANSACTION` and related attributes will be ignored.

Transaction Management for Context Service

By default, or by setting the value of the execution property `ManagedTask.TRANSACTION` to `ManagedTask.SUSPEND`:

- Any transaction that is currently active on the thread will be suspended.
- A `javax.transaction.UserTransaction` accessible in the local JNDI namespace as `java:comp/UserTransaction` will be available so that the contextual proxy object may begin, commit, and roll back a transaction.
- If a transaction begun by a contextual proxy object is not completed before the method ends, a `WARNING` will be logged in the output, and the transaction will be rolled back.

- The original transaction, if any, was active on the thread, it will be resumed when the task or contextual proxy object method returns.

By setting the value of the execution property `ManagedTask.TRANSACTION` to `ManagedTask.USE_TRANSACTION_OF_EXECUTION_THREAD`:

- The transaction will be managed by the execution thread and the task itself, so that any transaction that is currently active on the thread will not be suspended when the contextual proxy object method begins, and will not be resumed when the contextual proxy object method returns.
- If there is a currently active transaction on the thread, any resources used by the contextual proxy object will be enlisted to that transaction.
- If a transaction begun by the contextual proxy object is not completed before the method ends, the WebLogic Server will do nothing about it because there is the possibility that the transaction is completed by another method of the contextual proxy object.

Transaction Management for MTF

When using MTF, the transactions are managed as follows:

- The task runs without an explicit transaction (they do not enlist in the application component's transaction), so the `UserTransaction.getStatus()` method always returns `Status.STATUS_NO_TRANSACTION`, unless a new transaction is started in the task.
- If the transaction is not completed before the task method ends, a `WARNING` will be logged in the output, and the transaction will be rolled back.

Global CMO Templates

In addition to the JSR 236 default CMOs, you can also define global CMOs as templates in the domain's configuration by using the WebLogic Remote Console and the configuration MBeans.

CMOs specified in the `config.xml` can be assigned to any application or application component in the domain.

> **Note:**
>
> You should typically use the Remote Console to configure WebLogic Server's manageable objects and services and allow WebLogic Server to maintain the `config.xml` file.

You can define three types of CMO templates in a domain:

- Managed Executor Service Template
- Managed Scheduled Executor Service Template
- Managed Thread Factory Template

For example, if you define a `managed-executor-service-template`, a unique MES instance is created for each application deployed in the domain.

- Configuring CMO Templates using the Remote Console
- Using MBeans to Configure CMO Templates

Configuring CMO Templates using the Remote Console

You can create and configure CMO templates globally in the domain's configuration using the WebLogic Remote Console.

In the **Edit Tree**, go to **Scheduling**, then:

- **Managed Executor Service Templates**
- **Managed Scheduled Executor Service Templates**
- **Managed Thread Factory Templates**

See Concurrent Managed Object Templates in the *Oracle WebLogic Remote Console Online Help*.

Using MBeans to Configure CMO Templates

CMO templates can be configured using the following configuration MBeans under the `DomainMBean`.

- ManagedExecutorServiceTemplateMBean
- ManagedScheduledExecutorServiceTemplateMBean
- ManagedThreadFactoryTemplateMBean

For more information, see the Domain Configuration MBeans section in the *MBean Reference for Oracle WebLogic Server*.

Configuring Concurrent Constraints

Constraints can also be defined globally in the domain's configuration using the WebLogic Remote Console and configuration MBeans. Concurrent constraints specified in the `config.xml` can be assigned to any application or application component in the domain.

- Using the Remote Console to Configure Concurrent Constraints
- Using MBeans to Configure Concurrent Constraints

Using the Remote Console to Configure Concurrent Constraints

Concurrent constraints can be configured in the domain configuration, in specified server instances and in server templates for dynamic clusters, using the Remote Console.

Domain-level Concurrent Constraints

To configure concurrent constraints for a domain, in the **Edit Tree**, go to the **Environment**: **Domain**: **Concurrency** page. Retain or update the default values in the **Max Concurrent New Threads** and **Max Concurrent Long Requests** fields.

Server-level Concurrent Constraints

To configure concurrent constraints for a specific server instance in a domain, in the **Edit Tree**, go to the **Environment**: **Servers**: *myServer*: **Advanced**: **Concurrency** page. Retain or update the default values in the **Max Concurrent New Threads** and **Max Concurrent Long Requests** fields.

Dynamic Cluster-level Concurrent Constraints

To configure concurrent constraints for server templates in a dynamic cluster, in the **Edit Tree**, go to the **Environment**: **Server Templates**: *myServerTemplate*: **Advanced**: **Concurrency** page. Retain or update the default values in the **Max Concurrent New Threads** and **Max Concurrent Long Requests** fields.

Using MBeans to Configure Concurrent Constraints

Concurrent constraints can be configured globally in the domain's configuration, in specified server instances, and in server templates for dynamic clusters using the following methods under the `DomainMBean`, `ServerMBean`, and `ServerTemplateMBean`:

- `maxConcurrentLongRunningRequests()` – See Setting Limits for Maximum Concurrent Long Running Requests.
- `maxConcurrentNewThreads()` – See Setting Limits for Maximum Concurrent New Threads.

For more information about using WebLogic Server MBeans, see Accessing WebLogic Server MBeans with JMX in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.

Querying CMOs

You can query global CMOs using administrative tools such as the Remote Console and MBeans:

- Using MBeans to Monitor CMOs
- Using MBeans to Monitor Concurrent Constraints

Using MBeans to Monitor CMOs

CMOs can be monitored using the following runtime MBeans under the `DomainMBean`.

- `ManagedExecutorServiceRuntimeMBean`

  The `ManagedExecutorServiceRuntimeMBean` can be accessed from the following MBean attributes:

  - `ApplicationRuntimeMBean.ManagedExecutorServiceRuntimes` – Provides statistics for all the Managed Executor Services of that application.
  - `ComponentRuntimeMBean.ManagedExecutorServiceRuntimes` – Provides statistics for all the Managed Executor Services of that module.

  See ManagedExecutorServiceTemplateMBean in *MBean Reference for Oracle WebLogic Server*.

- `ManagedScheduledExecutorServiceRuntimeMBean`

  The `ManagedScheduledExecutorServiceRuntimeMBean` can be accessed from the following MBean attributes:

  - `ApplicationRuntimeMBean.ManagedScheduledExecutorServiceRuntimes` – Provides statistics for all the Managed Scheduled Executor Services of that application.
  - `ComponentRuntimeMBean.ManagedScheduledExecutorServiceRuntimes` – Provides statistics for all the Managed Scheduled Executor Services of that module.

See ManagedScheduledExecutorServiceRuntimeMBean in *MBean Reference for Oracle WebLogic Server*.

- `ManagedThreadFactoryRuntimeMBean`

  The `ManagedThreadFactoryRuntimeMBean` can be accessed from the following MBean attributes:

  - `ApplicationRuntimeMBean.ManagedThreadFactoryRuntimes` – Provides statistics for all the Managed Thread Factories of that application.
  - `ComponentRuntimeMBean.ManagedThreadFactoryRuntimes` – Provides statistics for all the Managed Thread Factories of that module.

  See ManagedThreadFactoryRuntimeMBean in *MBean Reference for Oracle WebLogic Server*.

See Accessing WebLogic Server MBeans with JMX in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.

Using MBeans to Monitor Concurrent Constraints

A server's concurrent constraints can be monitored using the `ConcurrentManagedObjectsRuntimeMBean`, which can be accessed from the following MBean attribute:

- `ServerRuntimeMBean.ConcurrentManagedObjectsRuntime` – Provides statistics for threads created by concurrent managed objects of global runtime.

See ConcurrentManagedObjectsRuntimeMBean in *MBean Reference for Oracle WebLogic Server*.

See Accessing WebLogic Server MBeans with JMX in *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.

6

Using the Batch Runtime

WebLogic Server implements the batch runtime to provide support for defining, implementing, and running batch jobs, as defined for Jakarta Batch 1.0.

- About Batch Jobs
  Batch jobs are tasks that can be executed without user interaction and are best suited for non-interactive, bulk-oriented and long-running tasks that are resource intensive, can execute sequentially or parallel, and may be initiated ad hoc or through scheduling.
- Using the Default Batch Runtime Configuration with the Derby Database
  Batch applications can be deployed and started on WebLogic Server out-of-the-box with no runtime configuration. This is useful for smaller development environments that do not process and store large amounts of data.
- Configuring the Batch Runtime to Use a Dedicated Database
  The batch runtime in WebLogic Server uses an XA-capable data source to access the JobRepository tables for batch jobs and a managed executor service to execute asynchronous batch jobs. The managed executor service processes the jobs and the JobRepository data source stores the status of current and past jobs.
- Querying the Batch Runtime
  You can query the batch runtime's JobRepository for domain scope using MBeans.
- Troubleshooting Tips
  Learn tips for configuring and using the batch runtime with WebLogic Server.

About Batch Jobs

Batch jobs are tasks that can be executed without user interaction and are best suited for non-interactive, bulk-oriented and long-running tasks that are resource intensive, can execute sequentially or parallel, and may be initiated ad hoc or through scheduling.

As described in the Java EE 7 Tutorial, the batch framework consists of:

- A job specification language based on XML.
- A set of batch annotations and interfaces for application classes that implement the business logic.
- A batch container that manages the execution of batch jobs.
- Supporting classes and interfaces to interact with the batch container.

For detailed information about batch jobs, batch processing, and the batch processing framework, see Batch Processing in *The Java EE 7 Tutorial*. Also, see Jakarta Batch 1.0. The specification defines the programming model for batch applications and the runtime for scheduling and executing batch jobs.

- Use of Multiple Batch Runtime Instances
- Batch 1.0 Code Examples in WebLogic Server

Use of Multiple Batch Runtime Instances

WebLogic Server supports the ability for multiple batch runtime instances to run in a domain, whereby each instance is hosted on an individual Managed Server instance. However, there is no state replication across batch jobs running in a domain; that is, one batch runtime instance cannot be aware of another. Consequently, the processing for a given batch job occurs only on the batch runtime instance that is hosted on a single Managed Server. Once a batch job is started on a Managed Server instance, the job runs on that instance to completion. Any step in the job that is started, including any concurrent steps, runs only on that Managed Server instance. This behavior has important implications in both clustered and nonclustered environments, particularly with regards to load balancing, as follows:

- If your deployed applications allow external users or processes to access those applications to start batch jobs, the load balancing mechanism for distributing work across the Managed Servers instances typically ensures that the workload on each batch runtime instance, over time and on average, is similarly balanced.
- If you have an application that contains a batch job, and the application is replicated on multiple nonclustered Managed Server instances, you must create a data source on each Managed Server that points to the same database. You then configure each batch runtime instance to use that database for the job repository. In this manner, an incoming request that is routed by the load balancer can land on any Managed Server instance and start a new batch job.
- Although batch job processing cannot be clustered, batch applications can be deployed to a cluster. You can do this by creating a data source and targeting it to the cluster. Then you configure a group of batch runtime instances, each running in a Managed Server instance in the cluster, to use the same job repository.

  For more information about batch processing in a clustered environment, see Batch Applications in *Administering Clusters for Oracle WebLogic Server*.
- There is no guarantee that the batch load at any point in time is equivalent across the Managed Server instances if different requests can generate different types of batch load. The likelihood of uneven load distribution increases if there is high degree of variability of the types of batch load that can be generated by different requests. For example, if one request is sent to one batch runtime instance, and a second request is sent to the other batch runtime instance, it is possible that the first request could start 10 batch jobs and the second request start only 2 jobs. In this scenario, it is possible for the batch job workload to become unevenly distributed.

Batch 1.0 Code Examples in WebLogic Server

When you install the WebLogic Server code examples, the examples source code is placed in the *EXAMPLES_HOME*\examples\src\examples directory. The default path is *ORACLE_HOME*\wlserver\samples\server. From this directory, you can access the source code and instruction files for the Batch 1.0 examples without having to set up the samples domain.

The *ORACLE_HOME*\user_projects\domains\wl_server directory contains the WebLogic Server examples domain; it contains your applications and the XML configuration files that define how your applications and Oracle WebLogic Server will behave, as well as startup and environment scripts. See Sample Applications and Code Examples in *Understanding Oracle WebLogic Server*.

- **Using the Batch Job Operator** – demonstrates how to use the javax.batch.operations.JobOperator interface to submit batch jobs. The JobOperator interface provides a set of operations to start, stop, restart, and inspect jobs. This sample will also demonstrates how to use listeners to notify about specific event occurring during the batch processing execution.

  EXAMPLES_HOME/examples/src/examples/javaee7/batch/joboperator-api

- **Using Batch Parallelization Model to Run Partitioned Job Steps** – demonstrates how to use the `PartitionMapper` interface to enable finer control over parallel processing.

  EXAMPLES_HOME`/examples/src/examples/javaee7/batch/partition`

- **Avitek Medical Records (MedRec)** – A comprehensive educational sample application that demonstrates WebLogic Server and Jakarta EE features, as well as best practices. For Java EE 7, MedRec showcases batch processing's capability by compiling drug statistics in the background for the administrator. The statistics sum up the cost by record, physician and drug perspectives with a start date and end date, altogether in one batch, but with three outputs.

  Avitek Medical Records is optionally installed with the WebLogic Server code examples. You can start MedRec from the *ORACLE_HOME*`/user_projects/domains/medrec` directory, where *ORACLE_HOME* is the directory you specified as the Oracle Home when you installed Oracle WebLogic Server.

Oracle recommends that you run these examples before programming your own applications that use batch.

Using the Default Batch Runtime Configuration with the Derby Database

Batch applications can be deployed and started on WebLogic Server out-of-the-box with no runtime configuration. This is useful for smaller development environments that do not process and store large amounts of data.

When no batch runtime configuration exists, WebLogic Server uses:

- The demo Derby database to create a data source needed to update the job repository to persist batch job details.
- The executor service that is bound to default JNDI name of `java:comp/DefaultManagedExecutorService` (as required by the Java EE 7 specification).

In orders to access the default batch runtime configuration, WebLogic Server must be started using the `startWeblogic.sh` script.

See Querying the Batch Runtime.

Configuring the Batch Runtime to Use a Dedicated Database

The batch runtime in WebLogic Server uses an XA-capable data source to access the JobRepository tables for batch jobs and a managed executor service to execute asynchronous batch jobs. The managed executor service processes the jobs and the JobRepository data source stores the status of current and past jobs.

The default batch runtime in a WebLogic domain can be used without any configuration, which is useful in development mode environments that only require the Derby demo database. For data-driven production environments that use a database schema, you can configure a dedicated job repository data source and managed executor service for the domain.

- Prerequisite Steps: Configure the Job Repository Tables, Batch Data Source, and Managed Executor Service
- Configure the Batch Runtime to Use a Dedicated Batch Data Source and Managed Executor Service

Prerequisite Steps: Configure the Job Repository Tables, Batch Data Source, and Managed Executor Service

For enterprise-level production environments that process and store large amounts of data, a dedicated batch runtime can be configured to store the batch job details in a specific database.

- Create the Job Repository Tables
- Create a JDBC Data Source for the Job Repository
- Optionally, Create a Managed Executor Service Template

Create the Job Repository Tables

The database administrator must create the job repository tables needed to persist batch job details. The schema name used to created these tables will be denoted by the `getSchemaName()` method in the `BatchConfigMBean` when configuring the batch runtime for the domain. See Configure the Batch Runtime to Use a Dedicated Batch Data Source and Managed Executor Service.

The job repository tables can be created using the Repository Create Utility (RCU) or using SQL scripts for the databases supported for use with WebLogic Server 14c. Schemas for creating these tables are in the following location:

ORACLE_HOME`/oracle_common/common/sql/wlservices/batch/`*dbname*

Iwhere *ORACLE_HOME* represents the top level installation directory for Oracle WebLogic Server, and *dbname* represents the name of the database.

For information about the supported databases for WebLogic Server 14c, see the Oracle Fusion Middleware Supported System Configurations page on Oracle Technology Network.

- Creating Job Repository Tables Using RCU
- Creating Job Repository Tables Using an SQL Script

Creating Job Repository Tables Using RCU

It is important to note the following when using RCU to create the job repository tables and schema owner:

1. On the **Select Components** page, select **WebLogic Services** as the component. Also, note that **Schema Owner** name will default to the schema prefix string you chose plus "WLS", such as `JBatch_WLS`. Write this name down because you will use it when you create the batch data source and the batch runtime.
2. On the **Schema Passwords** page, choose the **Select same password for all schemas** option.
3. When you click **Finish**, RCU will create tables and schemas for all WebLogic related components, including Batch, EJB Timers, Diagnostics, etc.

Now you can create the batch data source, as described in Create a JDBC Data Source for the Job Repository. Remember that you must use the schema owner you chose on the **Select Components** page as the data source's user name.

> **Note:**
>
> The batch runtime caches the schema name and when it acquires a connection to update the job repository tables, it sets the schema name on the connection. Due to this limitation, it is not possible to use the same data source for both application and job repository (if they use separate schemas).

For more information about the Repository Clean Utility (RCU), see *Creating Schemas with the Repository Creation Utility*.

Creating Job Repository Tables Using an SQL Script

If you are not using RCU utility to create the job repository tables for batch, you can use the SQL command-line utility and the provided `batch.sql` script to create them. For example, when you create job repository tables for Oracle Database and Oracle EBR (Edition-Based Redefinition), which require SQL to create the tables.

The `batch.sql` SQL script is provided for all supported databases (such as, mysql, db2, etc.) to create the job repository tables, and are in the following location:

ORACLE_HOME`/oracle_common/common/sql/wlservices/batch/`*dbname*

To use the `batch.sql` SQL script to create the tables, follow these steps:

1. Open an SQL command-line session for your database.
2. Create a new user called `jbatch` that will be identified by the `batch.sql` script.
3. Grant `Connect` privileges to user `jbatch`.
4. Grant `Resource` privileges to user `jbatch`.
5. Run the `batch.sql` script from the directory containing the Oracle database's SQL scripts. For example:

   ORACLE_HOME`/oracle_common/common/sql/wlservices/batch/oracle/batch.sql`

Now you can create the batch data source, as described in Create a JDBC Data Source for the Job Repository. Remember that you must use the `jbatch` schema owner you created as the data source's user name.

Create a JDBC Data Source for the Job Repository

For a dedicated batch runtime within a domain, the WebLogic administrator must configure an XA-capable data source for the database that will contain the job repository tables. When a Jakarta EE component submits a batch job, the batch runtime updates the job repository tables using this XA data source, which is obtained by looking up the data source's JNDI name.

When you create the batch data source using WLST, you must use the schema owner created with RCU (e.g., `jbatch_wls`) or the SQL script `jbatch`, as described in Create the Job Repository Tables.

For instructions on configuring JDBC data source, see Creating a JDBC Data Source in *Administering JDBC Data Sources for Oracle WebLogic Server*.

Optionally, Create a Managed Executor Service Template

For optimum performance, the batch runtime can be configured to use application-scoped Managed Executor Services by configuring Managed Executor Service Templates (MES Template) that use the same name as the batch runtime `setBatchJobsManagedExecutorServiceName()`. If no MES Template is specified when configuring the batch runtime, it will instead use the default Jakarta Managed Executor Service that is bound to (`java:comp/DefaultManagedExecutorService`).

When a new instance of a Managed Executor Service is created for each MES template, it will then run batch jobs that are submitted for applications that are deployed to the domain. For example, if there are two MES Templates named `MES1` and `MES2` in a domain, then when `BatchApp1` and `BatchApp2` are deployed, each application will get an instance of `MES1` and `MES2`.

However, if you have set the `setBatchJobsExecutorServiceName("MES2")`, then all batch jobs submitted from `BatchApp1` or `BatchApp1` (or from any application deployed to the domain), will use `MES2`.

For instructions on configuring a Managed Executor Service Template, see Configuring Concurrent Managed Objects.

Configure the Batch Runtime to Use a Dedicated Batch Data Source and Managed Executor Service

The job repository data source and Managed Executor Service you created in Prerequisite Steps: Configure the Job Repository Tables, Batch Data Source, and Managed Executor Service can now be used to configure a dedicated batch runtime using any of these WebLogic administrative tools:

> **Tip:**
>
> The schema name used in Create the Job Repository Tables must be specified when following the configuration steps in these sections. For example when using MBeans, the schema name must be denoted by `getSchemaName()` in the `BatchConfigMBean` for the domain.

- Configuring the Batch Runtime Using WLST

Configuring the Batch Runtime Using WLST

You can use WLST with the `BatchRuntimeConfigMBean` and `DomainMBean` to configure the batch runtime to use a specific database for the job repository:

```
def update_domain_batch_config(domainName, jndiName, schemaName):
    connect('admin','passwd')
    edit()
    startEdit()
    cmo.setDataSourceJndiName(jndiName)
    cd('/BatchConfig/' + domainName)
    cmo.setSchemaName(schemaName)
    save()
    activate()
```

In this example, if the administrator has created a data source with the JNDI name `jdbc/batchDS`, then, calling `update_domain_batch_config('mydomain','jdbc/batchDS','BATCH')` will configure the batch runtime to store all the job repository tables in the schema `'BATCH'` in the database that is pointed by the data source that is bound to the `jndiName: 'jdbc/batchDS'`.

You can use WLST to configure the batch runtime to use specific Managed Executor Services for batch job execution. However, you must first create an Managed Executor Service and the name of the Managed Executor Service must be provided to the `DomainMBean`.

```
connect('admin','passwd')
 edit()
 startEdit()
 cmo.setBatchJobsExecutorServiceName('mesName')
 save()
 activate()
```

where *mesName* is the name of the Managed Executor Service that has already been created (and targeted) to this domain.

The batch runtime can be configured to use different Managed Executor Services using the `getBatchJobsManagedExecutorServiceName()` method in the `DomainMBean`. However, a Managed Executor Service Template by the same name must exist and be targeted to the domain scope when a batch job is submitted.

See the BatchConfigMBean and DomainMBean in the *MBean Reference for Oracle WebLogic Server*.

See WebLogic Server WLST Online and Offline Command Reference in the *WLST Command Reference for Oracle WebLogic Server*.

Querying the Batch Runtime

You can query the batch runtime's JobRepository for domain scope using MBeans.

> **Note:**
>
> Make sure that the database that contains the batch job repository is running. For example, the default Derby database is not automatically started when you boot WebLogic Server using the `java weblogic.Server` command. If your database is not running, an exception will be thrown by the Batch RI when you submit a batch job or when you access the BatchJobRepositoryRuntimeMBean using WLST. See Troubleshooting Tips.

- Using Runtime MBeans to Query the Batch Runtime

Using Runtime MBeans to Query the Batch Runtime

The job repository can be queried using WLST using the `BatchJobRepositoryRuntimeMBean` to obtain details about batch jobs in a domain.

See BatchJobRepositoryRuntimeMBean in the *MBean Reference for Oracle WebLogic Server*.

- Get Details of all Batch Jobs Using getJobDetails
- Get Details of a Job Execution Using getJobExecutions

- Get Details of a Job Step Execution Using getStepExecutions

Get Details of all Batch Jobs Using `getJobDetails`

The `getJobDetails()` attribute returns details about all the jobs submitted by applications deployed to the domain. Each entry in the collection contains an array of the elements.

Table 6-1 describes the elements in the `getJobDetails` attribute in the BatchJobRepositoryRuntimeMBean.

Table 6-1 Elements in getJobDetails() Attribute

Element Name	Description
JOB_NAME	The name of the batch job.
APP_NAME	The name of the application that submitted the batch job (String).
INSTANCE_ID	The instance ID (long).
EXECUTION_ID	The execution ID (long).
BATCH_STATUS	The batch status of this job (String).
START_TIME	The start time of the job (`java.util.Date`).
END_TIME	The completion time of the job (`java.util.Date`).
EXIT_STATUS	The exit status of the job (String).

Here is an example of a WLST script that uses `getJobDetails()` to print a list of all batch jobs deployed in a domain.

```
connect('admin', 'admin123')
domainRuntime()
cd('BatchJobRepositoryRuntime')
cd('myserver')
executions=cmo.getJobDetails(6)
print "JobName  AppName   InstanceID  ExecutionID  BatchStatus  StartTime  EndTime  ExitStatus"
 print e[0], " ", e[1], " ", e[2], "   ", e[3], "  ", e[4], "  ", e[5], " ", e[6],   ",e[7]
```

Here is sample output after running `getJobDetails()`:

```
JobName   InstanceID  ExecutionID   BatchStatus       StartTime
EndTime                   ExitStatus
PayrollJob     6           6         COMPLETED   Fri Apr 24 10:11:00 PDT 2015   Fri Apr 24 10:11:01 PDT
2015   COMPLETED
PayrollJob     5           5         COMPLETED   Fri Apr 24 10:10:57 PDT 2015   Fri Apr 24 10:10:58 PDT
2015   COMPLETED
PayrollJob     4           4         COMPLETED   Fri Apr 24 10:10:56 PDT 2015   Fri Apr 24 10:10:56 PDT
2015   COMPLETED
PayrollJob     3           3         COMPLETED   Mon Apr 20 11:32:12 PDT 2015   Mon Apr 20 11:32:12 PDT
2015   COMPLETED
PayrollJob     2           2         COMPLETED   Mon Apr 20 11:32:10 PDT 2015   Mon Apr 20 11:32:11 PDT
2015   COMPLETED
PayrollJob     1           1         COMPLETED   Mon Apr 20 11:25:26 PDT 2015   Mon Apr 20 11:25:26 PDT
2015   COMPLETED
```

Get Details of a Job Execution Using `getJobExecutions`

The `getJobExections` attribute returns details about a particular job execution. Each entry in the collection contains an array of the elements.

Table 6-2 describes the elements in the `getJobExecutions()` attribute of the BatchJobRepositoryRuntimeMBean.

Table 6-2 Elements in getJobExecutions() Attribute

Element Name	Description
JOB_NAME	The name of the batch job (String).
INSTANCE_ID	The instance ID (long).
EXECUTION_ID	The execution ID (long).
BATCH_STATUS	The batch status of this job (String).
START_TIME	The start time of the job (`java.util.Date`).
END_TIME	The completion time of the job (`java.util.Date`).
EXIT_STATUS	The exit status of the job (String).

Here is an example of using `getJobExectuions()` in a WLST script to get details for a given ExecutionID: `getJobExecutions(6)`. To get a list of all ExecutionIDs, use the `getJobDetails()` method.

```
connect('admin', 'admin123')
domainRuntime()
cd('BatchJobRepositoryRuntime')
cd('myserver')
executions=cmo.getJobExecutions(6)
print  "JobName    InstanceID   ExecutionID   BatchStatus   StartTime   EndTime    ExitStatus"
for e in executions
   print e[0], "  ", e[1], "    ", e[2], "    ", e[3], "    ", e[4], "  ", e[5], "  ", e[6]
```

Here is sample output after running `getJobExecutions()`:

```
JobName   InstanceID  ExecutionID   BatchStatus       StartTime
EndTime                   ExitStatus
PayrollJob     6           6         COMPLETED   Fri Apr 24 10:11:00 PDT 2015   Fri Apr 24 10:11:01 PDT
2015    COMPLETED
```

Get Details of a Job Step Execution Using `getStepExecutions`

The `getStepExecutions` attribute returns metrics about each step in a Job execution. Each entry in the collection contains an array of the elements.

Table 6-3 describes the elements in the `getStepExecutions()` attribute in the BatchJobRepositoryRuntimeMBean.

Table 6-3 Elements in getStepExecutions() Attribute

Element Name	Description
STEP_NAME	The name of the batch job step (String).
STEP_ID	The step ID (long).
EXECUTION_ID	The execution ID (long).
BATCH_STATUS	The batch status of this job (String).
START_TIME	The start time of the job (`java.util.Date`).
END_TIME	The completion time of the job (`java.util.Date`)

Table 6-3 (Cont.) Elements in getStepExecutions() Attribute

Element Name	Description
EXIT_STATUS	The exit status of the job (String).

Here is an example of using `getStepExecutions()` in a WLST script to get details for a given StepExecutionID: `getStepExecutions(6)`. To get a list of all ExecutionIDs, use the `getJobDetails()` method.

```
connect('admin', 'admin123')
domainRuntime()
cd('BatchJobRepositoryRuntime')
cd('myserver')
executions=cmo.getStepExecutions(6)
print  "StepName   StepExecutionID   BatchStatus    StartTime    EndTime    ExitStatus"
   print e[0], "     ", e[1], "      ", e[2], "     ", e[3], " ", e[4], "   ", e[5], "]
```

Here is sample output after running `getStepExecutions()`:

```
StepName   StepExecutionID  BatchStatus       StartTime                      EndTime
ExitStatus
PayrollJob     6           6         COMPLETED   Fri Apr 24 10:11:00 PDT 2015   Fri Apr 24 10:11:01 PDT
2015    COMPLETED
```

Troubleshooting Tips

Learn tips for configuring and using the batch runtime with WebLogic Server.

- Make Sure the Database Containing the Job Repository Tables is Running

Make Sure the Database Containing the Job Repository Tables is Running

A common mistake made by users of the Batch RI (reference implementation) is to neglect starting the database that contains the job repository tables. For example, if you boot WebLogic using the `java weblogic.Server` command, the Derby database is not automatically started. If the DB isn't running, then when you submit a batch job or access the `JobRepositoryRuntimeMBean` (using WLST), that job will fail and a cryptic exception will be thrown by the Batch RI:

```
[1] Exception thrown by Refernce Implementation (from IBM):

Caused by: weblogic.common.resourcepool.ResourceSystemException: Cannot load driver
class org.apache.derby.jdbc.ClientDataSource for datasource '<<<Data Source name>>>>'
```

Here is another error message that could be thrown by the Batch RI when the job repository's database isn't running:

```
Caused By: com.ibm.jbatch.container.exception.PersistenceException:
weblogic.jdbc.extensions.ConnectionDeadSQLException:
weblogic.common.resourcepool.ResourceDeadException:
0:weblogic.common.ResourceException: Could not create pool connection for datasource
'_com_oracle_weblogic_batch_connector@_com_oracle_weblogic_batch_connector_impl_@_com_ora
cle_weblogic_batch_connector_impl_WLSDatabaseConfigurationBean@_com_oracle_batch_internal
__derby_batch_DataSource'.
The DBMS driver exception was: java.net.ConnectException : Error connecting to server
localhost on port 1,527 with message Connection refused.
     at
```

```
com.ibm.jbatch.container.services.impl.JDBCPersistenceManagerImpl.getConnectionToDefaultS
chema(JDBCPersistenceManagerImpl.java:354)
     at
com.ibm.jbatch.container.services.impl.JDBCPersistenceManagerImpl.isDerby(JDBCPersistence
ManagerImpl.java:182)
     at
com.ibm.jbatch.container.services.impl.JDBCPersistenceManagerImpl.init(JDBCPersistenceMan
agerImpl.java:143)
     at
com.ibm.jbatch.container.servicesmanager.ServicesManagerImpl$ServiceLoader.getService(Ser
vicesManagerImpl.java:404)
     at
com.ibm.jbatch.container.servicesmanager.ServicesManagerImpl$ServiceLoader.access$300(Ser
vicesManagerImpl.java:388)
```

If you see these errors, make sure that the database that contains the batch runtime job repository is running.