BEA WebLogic Server®

Designing and Configuring WebLogic Server Environments

BETA

Version 9.0 BETA
Revised: December 15, 2004

Copyright

Copyright © 2004-2005 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, BEA WebLogic Server, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Enterprise Security, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic JRockit, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server Process Edition, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

BETA

Contents

BETA

BETA

BETA

6. Using Node Manager to Control Servers

BETA

BETA

7. Using the WebLogic Persistent Store

BETA

BETA

BETA

Introduction and Roadmap

BETA

This section describes the contents and organization of this guide—*Designing and Configuring WebLogic Server Environments*.

- "Document Scope and Audience" on page 1-1
- "Guide to This Document" on page 1-1
- "Related Documentation" on page 1-2
- "New and Changed Features in WebLogic Server Environments" on page 1-2

Document Scope and Audience

This document describes how you design, configure, and manage WebLogic Server® environments. It is a resource for system administrators and operators responsible for implementing a WebLogic Server installation. This document is relevant to all phases of a software project, from development through test and production phases.

It is assumed that the reader is familiar with J2EE and Web technologies, object-oriented programming techniques, and the Java programming language.

Guide to This Document

The document is organized as follows:

- This chapter, "Introduction and Roadmap," describes the scope of this guide and lists related documentation.

- Chapter 2, "Using Work Managers to Optimize Scheduled Work," describes the WebLogic Server execution model and the process of configuring application access to the execute queue.
- Chapter 3, "Avoiding and Managing Overload," describes detecting, avoiding, and recovering from overload conditions.
- Chapter 4, "Configuring Network Resources," describes optimizing your WebLogic Server domain for your network.
- Chapter 5, "Configuring Web Server Functionality," describes using WebLogic Server as a Web server.
- Chapter 6, "Using Node Manager to Control Servers," describes using Node Manager for the remote control of Administration and Managed Server instances.
- Chapter 7, "Using the WebLogic Persistent Store," describes configuring and monitoring the persistent store, a built-in, high-performance storage solution for WebLogic Server subsystems and services that require persistence.

Related Documentation

- *Understanding Domain Configuration*
- *Administration Console Online Help*

New and Changed Features in WebLogic Server Environments

The following sections describe key changes and improvements to WebLogic Server:

- "Server Self-Tuning for Production Environments" on page 1-2
- "New Overload Protection Increases Availability" on page 1-3
- "Network Channels Can Manage Traffic Between Server Instances" on page 1-3
- "Node Manager Enhancements" on page 1-4
- "System-Wide Persistent Store" on page 1-5

Server Self-Tuning for Production Environments

New self-tuning capabilities simplify the process of configuring WebLogic Server for production environments with service level requirements that vary over time or by application. Self-tuning

BETA

helps prevent deadlocks during periods of peak demand. Self-tuning features are also useful if your WebLogic Server environment hosts multiple applications with different performance and availability requirements—for example, allowing you to allocate a greater percentage of resources to a user-facing order processing application than to a back-end inventory management application.

The new queue strategy enables administrators to allocate processing resources and manage performance more effectively, by avoiding the effort and complexity involved in configuring, monitoring, and tuning custom executes queues.

Key self-tuning features in WebLogic Server include:

- Workload management—Administrators can define scheduling policies and constraints at the domain level, application level, and module level.
- Automatic thread count tuning—A thread pool can maximize throughput by automatically changing its size, based on throughput history and queue size.
- Thread scheduling functionality—WebLogic Server 9.0 implements the Java Connector Architecture (JCA) API, exposing thread scheduling functionality to developers. Applications can use the Work Manager API to execute work asynchronously and receive notifications on the execution status.

  For more information, see "Using Work Managers to Optimize Scheduled Work" on page 2-1.

BETA

New Overload Protection Increases Availability

WebLogic Server 9.0 has improved capabilities for detecting when system load increases to the point that application performance and stability might be at risk. These new features, referred to as overload protection, are key to avoiding, and minimizing the negative effects of overload.

New overload features protect a server instance from out-of-memory (OOM) exceptions, execute queue overloads, increasing the availability of a server or a cluster.

For more information, see "Avoiding and Managing Overload" on page 3-1.

Network Channels Can Manage Traffic Between Server Instances

In addition to managing external network traffic, network channels can now manage network traffic between server instances. Other new and improved configuration and control options for network channels include:

- SSL behaviors configurable on a per-channel basis.
- Dynamic configuration and starting of channels without re-booting the server instance.
- New capabilities for closing and restarting a channel.
- Replication channel for replication traffic among server instances in a WebLogic Server cluster.

For more information, see "Configuring Network Resources" on page 4-1.

Node Manager Enhancements

A number of enhancements make Node Manager more versatile and easier to use:

- Shell Script Node Manager—WebLogic Server 9.0 provides a version of Node Manager implemented as a shell script. The Node Manager shell script provides the same functionality as the Java Node Manager. It can be used with the Secure Shell (SSH) or Remote Shell (RSH) protocol for secure remote control of server instances running on UNIX or Linux systems.
- WebLogic Scripting Tool (WLST) support— You can now use WLST commands to access Node Manager and to start, stop, and suspend server instances remotely or locally; obtain server status; and retrieve the contents of the server output log, without requiring the presence of a running Administration Server. In addition, you can enroll the machine on which WLST is running to be monitored by Node Manager.

  For more information, see *WebLogic Scripting Tool*.
- Administration Server control—In previous versions, Node Manager required access to a running Administration Server, and could control and monitor only Managed Servers. In WebLogic Server 9.0, Node Manager can start, monitor, and restart Administration Servers.
- Node Manager and server migration—In WebLogic Server 9.0, Node Manager is used to accomplish migration of migratable servers in a WebLogic Server cluster.

  For more information, see "Server Migration" in *Using WebLogic Server Clusters*.
- Improved diagnostics and logging—Node Manager diagnostics are improved, and the logging strategy for Node Manager and the server instances it controls are simplified.
- Simplified setup—In WebLogic Server 9.0, Node Manager setup is simplified. In particular, Node Manager no longer requires two-way SSL. Only one-way SSL is required.

BETA

- Node Manager runs as a Windows service—BEA Systems recommends running Node Manager as an operating system service so that it is automatically restarted in the event of system failure or reboot and using Node Manager to start or restart servers.

  For more information, see "Installing the Node Manager as a Windows Service" in the *Installation Guide*.

System-Wide Persistent Store

The WebLogic Persistent Store is a built-in, high-performance storage solution for WebLogic Server subsystems and services that require persistence, especially subsystem that require the creation and deletion of short-lived data objects, such as transactional messages for JMS Servers. Each server instance in a domain has a default persistent store that requires no configuration and that can be used simultaneously by subsystems that do not require explicit selection of a particular store, but can use the system's default storage. These subsystems include JMS Servers, Web Services, EJB Timer services, Store-and-Forward services, and the JTA Transaction Log (TLOG). Optionally, administrators can configure dedicated file-based stores or JDBC-accessible stores to suit their environment.

For more information, see "Using the WebLogic Persistent Store" on page 7-1.

BETA

BETA

CHAPTER 2

Using Work Managers to Optimize Scheduled Work

BETA

WebLogic Server allows you to configure how your application prioitizes the execution of its work. 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 you application by defining a Work Manger and applying it either globally to WebLogic Server domain or to a specific application component.

Understanding How WebLogic Server Uses Thread Pools

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`.

Users 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.

In WebLogic Server 9.0 there is 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 9.0 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 application. At run-time, WebLogic Server uses these guidelines to assign pending work and enqueued requests to execution threads.

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:

For more information on these components, see "Request Classes" on page 2-4 or "Constraints" on page 2-6

BETA

You can use any of these components from to control your the performance of your application by referencing the name of the component in the application's deployment descriptor. In addition, you may define a *Work Manager* that encapsulates all of the above components (except Context Request Class. See "Context Request Class" on page 2-5) 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.

Work Managers can be configured at the domain level, application level, and module level in one of the following configuration files:

- `config.xml`—Work Managers specified in `config.xml` can be assigned to any application, or application component, in the domain. You can use the Administration Console to define a Work Manager.
- `weblogic-application.xml`—Work Managers specified in at the application level in can be assigned to that application, or any component of that application.
- `weblogic-ejb-jar.xml` or `weblogic.xml`—Work Managers specified in at the component-level can be assigned to that component.
- `weblogic-web-app.xml`—Work Managers specified for a Web Application.

Listing 2-1 is an example of a Work Manager definition.

Listing 2-1 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>
</work-manager>
```

To reference the Work Manager used in the example in Listing 2-1 in the dispatch policy of a Web Application, add the code in Listing 2-2 to the Web Application's `web.xml` file:

BETA

Listing 2-2 Referencing the Work Manager in a Web Application

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

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

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

There are multiple types of request class, 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 percentage of thread-use time used to process requests.

  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.

- `response-time-request-class`—This type of request class specifies a response time goal in milliseconds. Response time goals are not applied to individual requests. Instaead, 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 schedule requests so that the average wait for requests with the class is proportional to its tolerable waiting time.

  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 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

BETA

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.

- context-request-class—This type of 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 Listing assigns a request class to requests based on the value of the request's subject and role properties.

Context Request Class

A context request class allows you to define request classes in an application's deployment descriptor based on a user's context. For example:

Listing 2-3 Context Request Class

```
<work-manager>
  <name>responsetime_workmanager</name>
    <response-time-request-class>
      <name>my_response_time</name>
      <goal-ms>2000</goal-ms>
   </response-time-request-class>
</work-manager>

<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>
```

Above, we explained the 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.

BETA

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`—This 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.

  A `max-threads-constraint` can be defined in terms of a the availability of 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`—This constraint guarantees a 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 Then, however, it the constraint will cause WebLogic Server to schedule a request from a even if requests in the service class have gotten more than its fair share recently.

- `capacity`—This constrain causes the server to reject requests only when it has reached its capacity. The default is zero. 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.

BETA

Stuck Thread Handling

In response to stuck threads, you can define a Stuck Thread Work Manager component that can shut down the Work Manager, (this causes all threads to be killed) move the application into admin mode, or mark the server instance as failed. For instance, the Work Manager defined in Listing 2-4 shuts down the Work Manager when two threads are stuck for longer than 30 seconds.

Listing 2-4 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>
```

Assigning Work Managers to Applications and Application Components

Work Managers can be specified in the following descriptors:

- `config.xml`
- `weblogic-application.xml`
- `weblogic-ejb-jar.xml`
- `weblogic-web-app.xml`

If you do not assign a Work Manager to an application, it uses a 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 Listing 2-2, "Referencing the Work Manager in a Web Application," on page 2-4.

BETA

Using Work Managers, Request Classes, and Constraints

Work Managers, Request Classes, and Constraints require the following:

- A definition. You may define a Work Managers, Request Classes, or Constraints globally in the domain's configuration using the Administration Console, (see Environments >

Work Managers in the Administration Console) or you may define them in one of the deployment descriptors listed above. In either case, you assign a name to each.

- A mapping. In your deployment descriptors you reference on of the Work Managers, Request Classes, or Constraints by its name.

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, we allow `dispatch-policy`, `max-threads`, and `min-threads`, to specify named (or unnamed with 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

This section contains examples for defining Work Managers in various types of deployment descriptors.

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

- weblogic-ejb-jar.xml schema
- weblogic-application.xml schema
- weblogic-web-app.xml schema

Listing 2-5 weblogic-ejb-jar.xml With Work Manager Entries

```
<weblogic-ejb-jar xmlns="http://www.bea.com/ns/weblogic/90"
   xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.bea.com/ns/weblogic/90
   http://www.bea.com/ns/weblogic/90/weblogic-ejb-jar.xsd">
```

BETA

```
<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>
```

BETA

```
      <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>
```

Listing 2-6 weblogic-ejb-jar.xml with Connection Pool Based Max Thread Constraint

These EJBs 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.

```
</weblogic-ejb-jar>
<weblogic-ejb-jar xmlns="http://www.bea.com/ns/weblogic/90"
  .xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
 . xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 . xsi:schemaLocation="http://www.bea.com/ns/weblogic/90
 . http://www.bea.com/ns/weblogic/90/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>
```

BETA

```
<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>
```

Listing 2-7 weblogic-ejb-jar.xml with commonJ Work Managers

For information using commonJ, see the commonJ Javadocs.

Listing 2-8 weblogic-application.xml

```
<weblogic-application xmlns="http://www.bea.com/ns/weblogic/90"
   xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.bea.com/ns/weblogic/90
   http://www.bea.com/ns/weblogic/90/weblogic-application.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>
```

Listing 2-9 web application descriptor

This Web Application is deployed as part of the Enterprise Application defined in Listing 2-8, "weblogic-application.xml," on page 2-11. This Web Application's descriptor defines two Work Managers. Both Work Managers point to the same max threads constraint, `j2ee_maxthreads`

BETA

which is defined in the application's `weblogic-application.xml` file. Each Work Manager specifies a different response time request class.

```
<weblogic-web-app xmlns="http://www.bea.com/ns/weblogic/90"
   xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.bea.com/ns/weblogic/90
   http://www.bea.com/ns/weblogic/90/weblogic-web-app.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-web-app>
```

Listing 2-10 web application descriptor

This descriptor defines a Work Manager using the context-request-class.

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

<weblogic-web-app xmlns="http://www.bea.com/ns/weblogic/90"
xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/ns/weblogic/90
http://www.bea.com/ns/weblogic/90/weblogic-web-app.xsd">

 <work-manager>
  <name>foo-servlet-1</name>
  <request-class-name>test-fairshare2</request-class-name>
  <max-threads-constraint>
```

BETA

```
    <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>
    <request-class-name>test-fairshare2</request-class-name>
   </context-case>
  </context-request-class>
 </work-manager>
</weblogic-web-app>
```

BETA

BETA

Avoiding and Managing Overload

BETA

WebLogic Server 9.0 has new features for detecting, avoiding, and recovering from overload conditions. WebLogic Server's overload protection features help prevent the negative consequences—degraded application performance and stability—that can result from continuing to accept requests when the system capacity is reached.

- "Configuring WebLogic Server to Avoid Overload Conditions" on page 3-1
- "WebLogic Server Self-Monitoring" on page 3-4
- "WebLogic Server Exit Codes" on page 3-4

Configuring WebLogic Server to Avoid Overload Conditions

When system capacitiy is reached, if an application server continues to accept requests, application performance and stability can deteriorate. The following sections how you can configure WebLogic Server to minimize the negative results of system overload.

Limiting Requests in the Thread Pool

In WebLogic Server 9.0, 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 Max Open Socket Value 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 Max execute queue Length field in the Administration Console. (See Environments > Servers > Threads and select an execute queue.) The default value of this field is 65536.

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 global thread pool value. The limit is reached first is honored.

See "Using Work Managers to Optimize Scheduled Work" on page 2-1.

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 re-direct requests to alternative server instance.

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

BETA

- If the server instance is in a cluster, a 503 response is sent back. The behavior after this response is sent is determined by your load balancer or proxy server.

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 more minimize the impact of the out of memory condition—automatic shutdown helps avoid application instability, and you can configure Node Manager or another HA tool to automatically restart WebLogic Server, minimizing down-time.

Configure this behavior by checking the Exit server process on Out Of Memory Error option on the Server->Overload Protection page in the Administration Console.

Note: This Administration Console page is not implemented for the Beta release. You can enable this behavior by adding the following to the `config.xml` file for your domain:

```
<OverloadProtection ExitOnPanic="true"
</OverloadProtection>
```

For more information, see the attributes of the OverloadProtectionMBean.

BETA

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-part high-availability solution to restart the server instance for automatic failure recovery.

Configure a server instance to exit when all application threads are stuck by checking the Exit server process on detecting deadlock option on the Server->Overload Protection page in the Administration Console.

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.
- 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.

For more information, see the attributes of the OverloadProtectionMBean.

WebLogic Server Self-Monitoring

The following sections describe WebLogic Server features that aid in determining and reporting overload conditions.

Overloaded Health State

WebLogic Server 9.0 has a new 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 when Work Manager capacity is exceeded or as a result of low memory.

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 *Managing Server Startup and Shutdown*.

BETA

CHAPTER 4

Configuring Network Resources

BETA

Note: For a summary of new Networking features for WebLogic Server 9.0 Beta, see What's New in WebLogic Server 9.0.

Configurable WebLogic Server resources, including network channels and domain-wide administration ports, help you effectively utilize the network features of the machines that host your applications and manage quality of service.

The following sections describe configurable WebLogic Server network resources, examples of their use, and the configuration process:

- "Overview of Network Configuration" on page 4-1
- "Understanding Network Channels" on page 4-2
- "Configuring a Channel" on page 4-8

Overview of Network Configuration

For many development environments, configuring WebLogic Server network resources is simply a matter of identifying a Managed Server's 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 allows you to 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 timeout 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 with the Servers-->Protocols-->Channels tab of the Administration Console or by using `NetworkAccessPointMBean`.

New Network Configuration Features in WebLogic Server

In this version of WebLogic Server, the functionality of network channels is enhanced to simplify the configuration process. Network channels now encompass the features that, in WebLogic Server 7.x, required both network channels and network access points. In this version of WebLogic Server, network access points are deprecated. The use of `NetworkChannelMbean` is deprecated in favor of `NetworkAccessPointMBean.`

Understanding Network Channels

The sections that follow describe network channels and the standard channels that WebLogic Server pre-configures, and discusses 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 timeout value and maximum message sizes.

BETA

- 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.
- 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 WebLogic Server's default channel, which is automatically configured by WebLogic Server, based on the attributes in `ServerMBean` or `SSLMBean`. The default channel is described in "The Default Network Channel" on page 4-5.

`ServerMBean` and `SSLMBean` represent a server instance and its SSL configuration. When you configure a server instance's Listen Address, Listen Port, and SSL Listen port, using the Server-->Configuration-->General tab, 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 can 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:

BETA

- **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 onto different listen addresses or listen ports.

  You can also segregate instance administration and application traffic by configuring a domain-wide administration port or administration channel. For more information, see "Administration Port and Administrative Channel" on page 4-5.

- **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 vs. non-secure traffic.

- **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 multi-homed 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 multi-homed machine. See "The Default Network Channel" on page 4-5 for information on setting the listen address in `ServerMBean`.

BETA

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

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 communication (7001 by default), and one port for HTTPS communication (7002 by default). You can configure the Listen Address and Listen Port using the Configuration-->General tab in the Administration Console; 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.

BETA

Administration Port and Administrative Channel

You can define an optional administration port for your domain. When configured, the administration port is used by each Managed Server in the domain exclusively for communication with the domain's Administration Server.

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. For more information on the standby state, see "Standby State" in *Managing Server Startup and Shutdown*.

- Separate administration traffic from application traffic in your domain. In production environments, separating the two forms of 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 the `weblogic.Admin` command line utility. If you do not configure an administration port, administrative commands such as `THREAD_DUMP` and `SHUTDOWN` will not work on deadlocked server instances.

If a administration port is enabled, WebLogic Server automatically generates an administration channel based on the port settings upon server instance startup.

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's 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" on page 4-7.
- After enabling the administration port, all Administration 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 multi-homed machine and assign each server instance a unique listen address, or

BETA

– Override the domain-wide port on all but one of one of the servers instances on the machine. Override the port using the Local Administration Port Override option on the Advanced Attributes portion of the Server->Connections->SSL Ports page in the Administration 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 Server--> Configuration-->General tab in the Administration 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 Server-->Configuration-->General tab in the Administration 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 the SSL Protocol" in *Managing WebLogic Security*.

Configure Administration Port

Enable the administration port as described in "Enabling the Domain-Wide Administration Port" in *Administration 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:

BETA

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

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" on page 4-8, and select "admin" as the channel protocol. Note the configuration and usage guidelines described in:

- "Administration Port Requires SSL" on page 4-7
- "Booting Managed Servers to Use Administration Port" on page 4-7

Configuring a Channel

You can configure a network channel using Servers-->Protocols-->Channels tab in the Administration Console or using the `NetworkAccessPointMBean`.

For instructions to configure a channel for a non-clustered Managed Server, see "Configuring a Network Channel" in *Administration Console Online Help*. To configure a channel for clustered Managed Servers see, "Configuring Network Channels For a Cluster" on page 4-10.

For a summary of key facts about network channels, and guidelines related to their configuration, see "Configuring Channels: Facts and Rules" on page 4-8.

BETA

Configuring Channels: Facts and Rules

Follow these guidelines when configuring a channel.

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.

Configuration Changes are Not Dynamic

- After creating a new channel, restart the server instance for the channel settings to take effect. Similarly, you must restart the server instance for most channel configuration changes to take effect.

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 Advanced Options portion of Servers-->Protocols-->Channels tab in the Administration 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 *Using WebLogicServer Clusters*, 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 existence of firewall both channels must define `PublicAddress`, and they both must define the same `PublicAddress`.

BETA

Configuring Network Channels For a Cluster

To configure a channel for clustered Managed Servers, note the information in "Configuring Channels: Facts and Rules" on page 4-8, and follow the guidelines described in the following sections.

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 *Using WebLogic Clusters*, or
- Use the Administration Console to create a cluster in an existing domain, following the instructions "Configuring a Cluster" in *Administration Console Online Help*.

For information and guidelines about configuring a WebLogic Server cluster, see "Before You Start" in *Using WebLogic Clusters*.

BETA

Create and Assign the Network Channel

Use the instructions in "Configuring a Network Channel" in *Administration 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 identically, including its name, on each Managed Server in the cluster.
- 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 Server-->Protocols-->Channels-->Configuration tab.

  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, how WebLogic Server can dynamically generate the cluster address, see "Cluster Address" in *Using WebLogic Clusters*.

  Note: If you want to use dynamic cluster addressing, do not supply a cluster address on the Server-->Protocols-->Channels-->Configuration tab. If you supply a cluster address explicitly, that value will take precedence and WebLogic Server will not generate the cluster address dynamically.

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

You can assign 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`. For more information, see "`network-access-point`" in *Programming WebLogic Server EJBs*.

BETA

BETA

Configuring Web Server Functionality

BETA

The following sections discuss how to configure Web Server components for WebLogic Server:

Overview of Configuring Web Server Components

In addition to its ability to host dynamic Java-based distributed applications, WebLogic Server is also a fully functional Web server that can handle 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.

HTTP Parameters

You configure the HTTP operating parameters using the Administration Console for each Server instance or Virtual Host.

Table 5-1 HTTP Operating Parameters

Attribute	Description	Range of Values	Default Value
Post Timeout	This attribute sets the timeout (in seconds) that WebLogic Server waits between receiving chunks of data in an HTTP POST data. Used to prevent denial-of-service attacks that attempt to overload the server with POST data.	Integer	30
Max Post Size	Max Post Size (in bytes) for reading HTTP POST data in a servlet request. MaxPostSize < 0 means unlimited	Bytes	-1

Enable Keepalives	Whether or not HTTP keep-alive is enabled.	Boolean
		True = enabled
		False = not enabled
			True

Duration (labeled Keep Alive Secs on the Virtual Host panel)	Number of seconds to maintain HTTP Keep Alive before timing out the session.	Integer	30
HTTPS Duration (labeled Https Keep Alive Secs on the Virtual Host panel)	Number of seconds to maintain HTTPS Keep Alive before timing out the session.	Integer	60

BETA

Advanced Attributes

Table 5-2 Advanced Attributes

Attribute	Description	Range of Values	Default Value
Frontend Host	Set when the Host information coming from the URL may be inaccurate due to the presence of a firewall or proxy. If this parameter is set, the HOST header is ignored and this value is always used.	String	Null
Frontend HTTP Port	Set when the Port information coming from the URL may be inaccurate due to the presence of a firewall or proxy. If this parameter is set, the HOST header is ignored and this value is always used.	Integer	0
Frontend HTTPS Port	Set when the Port information coming from the URL may be inaccurate due to the presence of a firewall or proxy. If this parameter is set, the HOST header is ignored and this value is always used.	Integer	0

Send Server Header	If set to false, the server name is not sent with the HTTP response. Useful for wireless applications where there is limited space for headers.	Boolean
		True = enabled
		False = not enabled
			True

BETA

Table 5-2 Advanced Attributes

Attribute	Description	Range of Values	Default Value

Accept Context Path In Get Real Path	Beginning with the WebLogic Sever 8.1 release inclusion of the contextPath in the virtualPath to the context.getRealPath() will not be allowed as it breaks the case when the subdirectories have the same name as contextPath. In order to support applications which might have been developed according to the old behaviour BEA is providing a compatibility switch. This switch will be deprecated in future releases.	Boolean
		True = enabled
		False = not enabled
			False

BETA

Table 5-2 Advanced Attributes

Attribute	Description	Range of Values	Default Value
HTTP Max Message Size	Maximum HTTP message size allowable in a message header. This attribute attempts to prevent a denial of service attack whereby a caller attempts to force the server to allocate more memory than is available, thereby keeping the server from responding quickly to other requests. This setting only applies to connections that are initiated using one of the default ports (ServerMBean setListenPort and setAdministrationPort or SSLMBean setListenPort). Connections on additional ports are tuned via the NetworkChannelMBean.	Minimum*:* 4096 Maximum*:* 2000000000	-1

BETA

Table 5-2 Advanced Attributes

Attribute	Description	Range of Values	Default Value
HTTP Message Timeout	Maximum number of seconds spent waiting for a complete HTTP message to be received. This attribute helps guard against denial of service attacks in which a caller indicates that they will be sending a message of a certain size which they never finish sending. This setting only applies to connections that are initiated using one of the default ports (ServerMBean setListenPort and setAdministrationPort or SSLMBean setListenPort). Connections on additional ports are tuned via the NetworkChannelMBean.	Minimum: 0 Maximum: 480	-1

BETA

Configuring the Listen Port

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`.

You define a separate listen port for regular and secure (using SSL) requests. You define the regular listen port on the Servers node in the Administration Console, under the Configuration/General tab, and you define the SSL listen port under the Connections/SSL tab.

Configuring the Listen Ports from the Administration Console

1. In the left pane, expand the Servers folder.
2. Click the server whose Listen Port you want to configure.
3. In the right pane, click the Configuration tab and the General subtab.
4. If you want to disable the non-SSL listen port so that the server listens only on the SSL listen port, remove the checkmark from the Listen Port Enabled box.
5. If you are using the non-SSL listen port and you want to modify the default port number, change the default number in the Listen Port box.
6. If you want to disable the SSL listen port so that the server listens only on the non-SSL listen port, remove the checkmark from the Enable SSL Listen Port box.

   Note: You cannot disable both the non-SSL listen port and the SSL listen port. At least one port must be active.
7. If you want to modify the default SSL listen port number change the value in the SSL Listen Port box.
8. Click Apply.
9. Restart the server.

BETA

Web Applications

HTTP and Web Applications are deployed according to the Servlet 2.4 and JSP 2.0 specifications from Sun Microsystems, which describe the use of *Web Applications* as a standardized way of grouping together 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 normally use the name of the Web Application as part of the URI you use to request resources from the Web Application.

By default JSPs are compiled into the servers' temporary directory the location for which is: <domain_dir>\servers\.<server_dir>\tmp\<app-dir>

For more information, see Developing Web Applications, Servlets and JSPs for WebLogic Server.

Web Applications and Clustering

Web Applications can be deployed in a cluster of WebLogic Servers. When a user requests a resource from a Web Application, the request is routed to one of the servers of 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.

For more information, see Using WebLogic Server Clusters.

Designating a Default Web Application

Every server instance and virtual host in your domain can declare a *default Web Application*. The default Web Application responds to any HTTP request that cannot be resolved to another deployed Web Application. In contrast to all other Web Applications, the default Web Application does *not* use the Web Application name as part of the URI. Any Web Application targeted to a server or virtual host can be declared as the default Web Application. (Targeting a Web Application is discussed later in this section. For more information about virtual hosts, see "Configuring Virtual Hosting" on page 5-9).

The examples domain that is shipped with Weblogic Server has a default Web Application already configured. The default Web Application in this domain is named `DefaultWebApp` and is located in the `applications` directory of the domain.

If you declare a default Web Application that fails to deploy correctly, an error is logged and users attempting to access the failed default Web Application receive an HTTP `404` error message.

For example, if your Web Application is called `shopping`, you would use the following URL to access a JSP called `cart.jsp` from the Web Application:

```
http://host:port/shopping/cart.jsp
```

If, however, you declared `shopping` as the default Web Application, you would access `cart.jsp` with the following URL:

```
http://host:port/cart.jsp
```

(Where `host` is the host name of the machine running WebLogic Server and `port` is the port number where the WebLogic Server is listening for requests.)

To designate a default Web Application for a server or virtual host set the context root in the application.xml or weblogic.xml file to "".

BETA

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 Servers, clusters and Web Applications, depending on your application and Web server requirements.

Virtual hosting also allows you to create one directory to serve static files such as images for multiple Web Applications. For example, to serve image files using a virtual host, you can create a mapping similar to the folowing:

```
<virtual-directory-mapping>
  <local-path>c:/usr/gifs</local-path>
  <url-pattern>/images/*</url-pattern>
</virtual-directory-mapping>
```

A request to `HTTP:// localhost:7001/mywebapp/images/test.gif` will cause your WebLogic Server implementation to look for the requested image at: `c:/usr/gifs/images/*`.

This use of a virtual host requires you to create a directory named images to hold the image files. This directory must be located in the relative uri, such as "`/images/test.gif`".

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

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.

BETA

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
```

For more information, see "How WebLogic Server Resolves HTTP Requests" on page 5-11.

When using multiple Virtual Hosts with diferent 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.

BETA

Setting Up a Virtual Host

Use the Administration Console to define a virtual host.

1. Create a new Virtual Host.
   a. Expand the Services node in the left pane. The node expands and displays a list of services.
   b. Click the virtual hosts node. If any virtual hosts are defined, the node expands and displays a list of virtual hosts.
   c. Click Create a New Virtual Host in the right pane.
   d. Enter a name to represent this virtual host.
   e. Enter the virtual host names, one per line. Only requests matching one of these virtual host names will be handled by the WebLogic Server instance or cluster targeted by this virtual host.
   f. Click Create.
2. Define logging and HTTP parameters:

   a. (Optional) Click on the Logging tab and fill in HTTP access log attributes (For more information, see “Setting Up HTTP Access Logs” on page 5-14.)

   b. Select the HTTP tab and fill in the HTTP Parameters.

3. Define the servers that will respond to this virtual host.

   a. Select the Targets --> Servers tab. You will see a list of available servers.

   b. Select a server.

   c. Click Apply.

4. Define the clusters that will respond to this virtual host (optional). You must have previously defined a WebLogic Cluster. For more information, see Using WebLogic Server Clusters.

   a. Select the Targets tab.

   b. Select the Clusters tab. You will see a list of available clusters.

   c. Select a cluster.

   d. Click Apply.

5. Target Web Applications to the virtual host.

   a. Click the Web Applications node in the left panel.

   b. Select the Web Application you want to target.

   c. Select the Targets tab in the right panel.

   d. Select the Virtual Hosts tab.

   e. Select Virtual Host.

   f. Click Apply.

You must 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.

BETA

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. The examples below demonstrate 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. For more information, see Developing Web Applications, Servlets and JSPs for WebLogic Server.

The table below 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. You set Index Directories using the Administration Console, on the Web Applications node, under the Configuration →Files tab.

BETA

Table 5-3 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. For more information, see Configuring Servlets.

Table 5-3 Examples of How WebLogic Server Resolves URLs

URL	Index Directories Checked?	This file is served in response
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. For more information, see Configuring Servlets.
http://host:port/apples/pie.jsp	Does not matter	`pie.jsp`, from the top-level directory of the `apples` Web Application.
`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 For more information, see error-pageu.

BETA

Table 5-3 Examples of How WebLogic Server Resolves URLs

URL	Index Directories Checked?	This file is served in response
`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`.

* For more information, see Configuring Welcome Pages.

BETA

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, and follows a standard convention. 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 or for each virtual host that you define.

For information on setting up HTTP logging for a server or a virtual host, refer to the following topics in the Administration Console online help:

- Enabling and Configuring an HTTP Log
- Specifying HTTP Log File Settings for a Virtual Host

Log Rotation

You can also choose to rotate the log file based on either the size of the file or after a specified amount of time has passed. When either one of these two criteria are 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 inlcudes a numeric portion that is incremented upon each rotation. Separate HTTP Access logs are kept for each Web Server you have defined.

Common Log Format

The default format for logged HTTP information is the common log 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 "-"

BETA

Setting Up HTTP Access Logs by Using Extended Log Format

WebLogic Server also supports extended log file format, version 1.0, as defined by the W3C. This is an emerging standard, and WebLogic Server follows the draft specification from W3C. The current definitive reference may be found from the W3C Technical Reports and Publications page

The extended log format allows you to specify the type and order of information recorded about each HTTP communication. To enable the extended log format, set the Format attribute on the HTTP tab in the Administration Console to `Extended`. (See "Creating Custom Field Identifiers" on page 5-18).

You specify what information 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 # 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 legal directives at the head of the file.

BETA

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. Here is an 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:

These fields give the IP address and port of either the requesting client, or the responding server. This field has type <address>, as defined in the W3C specification. The supported prefixes 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. This field has type <name>, as defined in the W3C specification. The supported prefixes 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.

BETA

`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.

`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.

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. 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. A sample of the Java source for such a class is included in this document. See “Java Class for Creating a Custom ELF Field” on page 5-22.

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.

   For more information on the `Fields` directive, see “Creating the Fields Directive” on page 5-16.

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`

BETA

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" on page 5-19.
- Use the `FormatStringBuffer` class to create the contents of your custom field. Methods are provided to create suitable output. For more information on these methods, see theJavadocs for FormatStringBuffer.

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. For more information, see "Setting Up HTTP Access Logs by Using Extended Log Format" on page 5-16.

Note: When writing the Java class that defines your custom field, you should 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.

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`.

BETA

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

Table 5-4 Getter Methods of HttpAccountingInfo

`HttpAccountingInfo Methods`	**Where to find information on the methods**
`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.Http.ServletRequest
`String getContextPath();`	javax.servlet.http.Http.ServletRequest

BETA

Table 5-4 Getter Methods of HttpAccountingInfo

HttpAccountingInfo Methods	Where to find information on the methods
`Cookie[] getCookies();`	javax.servlet.http.Http.ServletRequest
`long getDateHeader(String name);`	javax.servlet.http.Http.ServletRequest
`String getHeader(String name);`	javax.servlet.http.Http.ServletRequest
`Enumeration getHeaderNames();`	javax.servlet.http.Http.ServletRequest
`Enumeration getHeaders(String name);`	javax.servlet.http.Http.ServletRequest
`int getIntHeader(String name);`	javax.servlet.http.Http.ServletRequest
`String getMethod();`	javax.servlet.http.Http.ServletRequest
`String getPathInfo();`	javax.servlet.http.Http.ServletRequest
`String getPathTranslated();`	javax.servlet.http.Http.ServletRequest
`String getQueryString();`	javax.servlet.http.Http.ServletRequest
`String getRemoteUser();`	javax.servlet.http.Http.ServletRequest
`String getRequestURI();`	javax.servlet.http.Http.ServletRequest
`String getRequestedSessionId();`	javax.servlet.http.Http.ServletRequest
`String getServletPath();`	javax.servlet.http.Http.ServletRequest
`Principal getUserPrincipal();`	javax.servlet.http.Http.ServletRequest
`boolean isRequestedSessionIdFromCookie();`	javax.servlet.http.Http.ServletRequest
`boolean isRequestedSessionIdFromURL();`	javax.servlet.http.Http.ServletRequest
`boolean isRequestedSessionIdFromUrl();`	javax.servlet.http.Http.ServletRequest
`boolean isRequestedSessionIdValid();`	javax.servlet.http.Http.ServletRequest
`String getFirstLine();`	Returns the first line of the HTTP request, for example: `GET /index.html HTTP/1.0`

BETA

Table 5-4 Getter Methods of HttpAccountingInfo

HttpAccountingInfo Methods	Where to find information on the methods
`long getInvokeTime();`	Returns the length of time it took for the service method of a servlet to write data back to the client.
`int getResponseStatusCode();`	javax.servlet.http.HttpServletResponse
`String getResponseHeader(String name);`	javax.servlet.http.HttpServletResponse

Listing 5-1 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"));
  }
}
```

BETA

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 console, under *Servers* or *virtual hosts*. If you define these attributes for a virtual host, the values set for the virtual host override those set under *Servers*.

`PostTimeoutSecs`

You can limit the amount of time that WebLogic Server waits between receiving chunks of data in an HTTP POST.

The default value for `PostTimeoutSecs` is 30.

`MaxPostTimeSecs`

Limits the total amount of 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 MaxPostTimeSecs.
```

The default value for `MaxPostTimeSecs` is 30.

`MaxPostSize`

Limits the number of bytes of data received in a POST from a single request. If this limit is triggered, a `MaxPostSizeExceeded` exception is thrown and the following message is sent to the server log:

```
POST size exceeded the parameter MaxPostSize.
```

An HTTP error code 413 (Request Entity Too Large) is sent back to the client.

If the client is in listening mode, it gets these messages. If the client is not in listening mode, the connection is broken.

The default value for `MaxPostSize` is -1.

BETA

Setting Up WebLogic Server for HTTP Tunneling

HTTP tunneling provides a way to simulate a stateful socket connection between WebLogic Server and a Java 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.

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 two attributes that you can configure in the Administration Console to tune a tunneled connection for performance. You access these attributes in the Servers section, under the Connections and Protocols tabs. 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`

When an HTTP tunnel connection is set up, the client automatically sends a request to the server, so that the server may volunteer a response to the client. The client may also include instructions in a request, but this behavior happens regardless of whether the client application needs to communicate with the server. If the server does not respond (as part of the application code) to the client request within the number of seconds set in this attribute, it does so anyway. The client accepts the response and automatically sends another request immediately.

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.

BETA

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 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.

You specify the listen port for WebLogic Server in the Administration Console under the "Servers" node, under the "Network" tab.

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.

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 `<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.

BETA

BETA

Using Node Manager to Control Servers

BETA

The server instances in a production WebLogic Server environment are often distributed across multiple machines and geographic locations.

Node Manager is a utility for remote control of WebLogic Server instances. Node Manager runs separately from WebLogic Server and allows you to perform common operations tasks for Administration Servers and Managed Servers. While use of Node Manager is optional, it provides benefits if your WebLogic Server environment hosts applications with high availability requirements.

Node Manager allows you to start and stop server instances—both Administration Servers and Managed Servers—remotely, and to monitor and automatically restart them after an unexpected failure.

The following sections describe Node Manager's features and operating environment.

- "Understanding Node Manager" on page 6-1
- "Setting Up Node Manager" on page 6-13
- "Starting and Stopping Node Manager" on page 6-23
- "Node Manager Log Files and Error Messages" on page 6-27

Understanding Node Manager

This section describes Node Manager features and capabilities.

New Node Manager Features in WebLogic Server 9.0

This section describes the enhancements to Node Manager in WebLogic Server 9.0:

- Shell Script Node Manager—WebLogic Server 9.0 provides a version of the Node Manager implemented as a shell script. The Node Manager shell script provides the same functionality as the Java Node Manager. It can be used with the Secure Shell (SSH) or Remote Shell (RSH) protocol for secure remote control of server instances running on UNIX or Linux systems.
- WLST Support—In WebLogic Server 9.0, Node Manager can be accessed using WebLogic Server Scripting Tool (WLST) commands.
- Administration Server Control—In previous versions, Node Manager required access to a running Administration Server, and could control and monitor only Managed Servers. In WebLogic Server 9.0, Node Manager can start, monitor, and restart an Administration Server.
- Node Manager and Server Migration—In WebLogic Server 9.0, Node Manager is used to accomplish migration of migratable servers in a WebLogic Server cluster. For more information, see "Server Migration" in *Using WebLogic Server Clusters*.
- Log Changes—Node Manager diagnostics are improved, and the logging strategy for Node Manager and the server instances it controls is simplified.
- Configuration Changes—In WebLogic Server 9.0, Node Manager setup is simplified. In particular, Node Manager no longer requires 2-way SSL. Only 1-way SSL is required.
- Use of start scripts—Node manager can start a remote server using a start script in which you can specify options for starting the server.

BETA

Node Manager Capabilities

Node Manager capabilities are described in the following sections. Node Manager must run on the machine that hosts the server instances it controls.

Start, Suspend, and Stop Administration Servers

You can use Node Manager to start, stop, and suspend an Administration Server. These Node Manager features can be accessed using WebLogic Scripting Tool (WLST) commands or scripts.

Start, Suspend, and Stop Managed Servers

You can use Node Manager to start, stop, and suspend a Managed Server or WebLogic Server cluster. These Node Manager features can be accessed using the Administration Console and WebLogic Scripting Tool (WLST) commands or scripts.

Note: When you start a Managed Server with Node Manager, the Administration Server for the domain must be available so that the Managed Server can obtain its configuration.

Note: Node Manager uses the same command arguments that you supply when starting a Managed Server using a script or at the command line. For information about startup arguments, see "weblogic.Server Command-Line Reference" in *WebLogic Server Command Reference*.

Node Manager starts a Managed Server in a dedicated process on the target machine, separate from the Node Manager and Administration Server processes, in the same directory where the Node Manager process is running. To run the Managed Server in a different directory, set the Root Directory attribute in the Server—>Configuration—>Server Start page.

Restart Administration Servers and Managed Servers

If a server instance that was started using Node Manager fails, Node Manager automatically restarts it. Node Manager determines whether a server instance it started has failed based on the server instance exit code.

The restart feature is configurable. Node Manager's default behavior is to:

- Automatically restart server instances under its control that fail. You can disable the feature, if desired.
- Restart failed server instances no more than a specified number of times. You define the maximum number of restarts by setting the `RestartMax` property, in the Node Manager `startup.properties` file.

Node Manager's ability to restart failed server instances is resilient to failure or restart of Node Manager itself. If Node Manager fails or is explicitly stopped, upon restart, it determines the server instances that were under its control when it exited, and can restart server instances, as necessary.

It is advisable to run Node Manager as an operating system service, so that it starts back up automatically if the machine it runs upon is restarted.

BETA

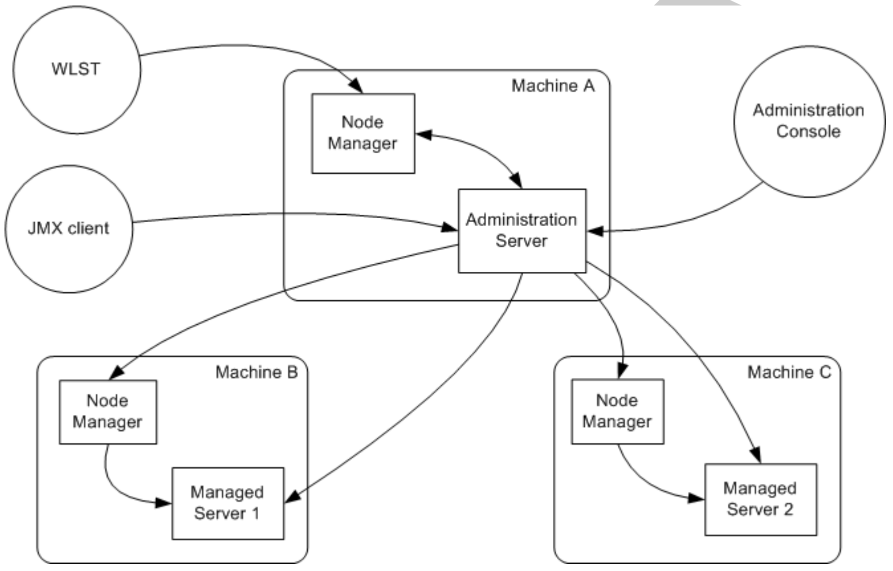
Monitor Servers and View Log Data

Node Manager creates a log file for the Node Manager process, and a log file of server output for each server instance it controls. You can view these log files, as well as regular log files for a server instance using the Administration Console or WLST commands. For more information, see "Node Manager Log Files and Error Messages" on page 6-27.

Node Manager Environment and Architecture

Figure 6-1 illustrates the relationship between Node Manager, its clients, and the server instances it controls.

Figure 6-1 Node Manager Architecture



The following sections describe key aspects of the Node Manager operating environment.

Node Manager Versions and Supported Operating Systems

WebLogic Server provides two versions of Node Manager:

- Java Node Manager—A Java server that runs as a Windows service for use on Windows or Unix systems. BEA provides native Node Manager libraries for Windows, Solaris, HP UX, Linux on Intel, Linux on Z-Series, and AIX operating systems.

Note: Native Node Manager is not supported on Open VMS, OS/390, AS400, UnixWare, or Tru64 UNIX.

- Node Manager Shell Script—New in WebLogic Server 9.0, a Unix shell script (`wlscontrol.sh` or `wlscontrol.cmd`), for use on Unix and Linux systems. The script is installed in `WL_HOME/common/nodemanager`.

You run the appropriate implementation of Node Manager on each system that hosts server instances. The Java server version and the shell script version are functionally equivalent.

Node Manager Security

Security is the main difference between the Java Node Manager and the SSH Node Manager.

Node Manager Shell Script Using SSH

The Node Manager SSH Shell Script relies on SSH user-based security to provide a secure trust relationship between users on different machines. Authentication is not required. You create a UNIX user account—typically one per domain—for running Node Manager commands and scripts. A user logged in as this user can issue Node Manager commands without providing a username and password. .

Java Node Manager Security

Clients connect to the Java Node Manager using 1-way SSL. (You can use plain text connections with Node Manager in a development environment.)

A user establishing a command line connection to the Java Node Manager using the WebLogic Server Scripting Tool (WLST) `nmConnect` command must provide the Node Manager user name and password. Node Manager verifies the username and password against the domain's `nm_password.properties` file.

Administration Console users do not need to explicitly provide credentials to connect to Node Manager—the Node Manager user name and password are available in the domain configuration and are provided automatically.

Note: For the WebLogic Server 9.0 Beta release, this only works if you create the managed server using the Configuration Wizard; if not, you specify the Node Manager username and password in the Console in the Domain—>Security—>General page, under the Advanced section.Then execute the `nmEnroll` command from the managed server using WLST.

BETA

Managed Server Remote Start Security

A remote start user name and password is required to start a server instance with Node Manager. These credentials are provided differently for Administration Servers and Managed Servers.

- Credentials for Managed Servers—When you invoke Node Manager to start a Managed Server it obtains its remote start name and password from the Administration Server.
- Credentials for Administration Servers—When you invoke Node Manager to start an Administration Server, the remote start user name can be provided on the command line, or obtained from the Administration Server's `boot.properties` file. The Configuration Wizard initializes the `boot.properties` file and the `startup.properties` file for an Administration Server when you create the domain.

Any server instance started by Node Manager encrypts and saves the credentials with which it started in a server-specific `boot.properties` file, for use in automatic restarts.

Node Manager Clients

You can access either version of Node Manager—the Java version or the SSH version—from these clients:

- Administration Server
  - using the Administration Console on the Server—>Control page.
  - using JMX, either the `weblogic.Admin` command-line utility (deprecated in this release of WebLogic Server) or utilities you write yourself.

    For more information about JMX, see *Developing Manageable Applications with JMX*.
- WLST commands and scripts—WLST offline serves as a Node Manager command-line interface that can run in the absence of a running Administration Server. You can use WLST commands to start, stop, and monitor a server instance without connecting to an Administration Server. Starting the Administration Server is the main purpose of the stand-alone client. However, you can also use it to:
  - Stop a server instance that was started by Node Manager.
  - Start a Managed Server.
  - Access the contents of a Node Manager log file.
  - Obtain server status.
  - Retrieve the contents of server output log.

BETA

- SSH Client—A shell command template is provided. for use with the SSH Node Manager.

A Node Manager client can be local or remote to the Node Managers with which it communicates.

Node Manager and Machines

Node Manager must run on each computer that hosts WebLogic Server instances that you want to control with Node Manager. Configure each computer as a Machine in WebLogic Server, and assign each server instance that you will control with Node Manager to the machine upon which it runs.

Node Manager should run as an operating system service or daemon, so that it is automatically restarted in the event of system failure or reboot. For more information, see "Installing the Node Manager as a Windows Service" in the *Installation Guide*.

Node Manager and Domains

A Node Manager process is not associated with a specific WebLogic domain. You can use the same Node Manager process to control server instances in any WebLogic Server domain, so long as they reside on the same machine as the Node Manager process.

Node Manager and Administration Servers

The relationship of an Administration Server to Node Manager varies for different scenarios.

- An Administration Server can be under Node Manager control—You can start it, monitor it, and restart it using Node Manager.
- An Administration Server can be a Node Manager client—When you start or stop Managed Servers from the Administration Console, you are accessing Node Manager via the Administration Server.
- An Administration Server supports the process of starting up a Managed Server with Node Manager—When you start a Managed Server with Node Manager, the Managed Server contacts the Administration Server to obtain outstanding configuration updates. For more information, see "Node Manager and Managed Servers" on page 6-7.

Node Manager and Managed Servers

Managed Servers are associated with Node Manager by assigning them to a Machine upon which Node Manager runs.

BETA

If you want Node Manager to be able to restart a Managed Server after failure even when the Administration Server is unavailable, Managed Server Independence (MSI) mode must be enabled for the Managed Server, as it is by default.

Note: Node Manager cannot start a Managed Server for the first time in MSI mode, because at first start up a Managed Server must be able to reach the Administration Server.

Node Manager Processes and Communications

The sections that follow describe how Node Manager accomplishes key functions,

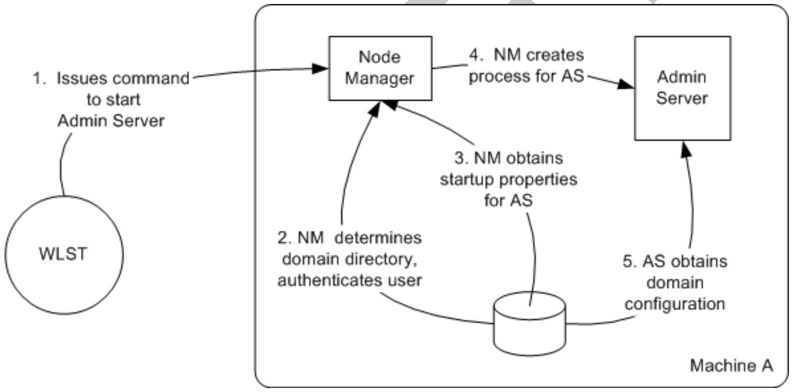
Starting an Administration Server With Node Manager

Figure 6-2 illustrates the process of starting an Administration Server with Node Manager.

This section assumes that you have installed the Administration Server and created its domain directory using the Configuration Wizard.

Node Manager is running on Machine A, which hosts the Administration Server. The stand-alone Node Manager client is remote.

Figure 6-2 Starting Administration Server


1. Issues command to start Admin Server
WLST
Node Manager
4. NM creates process for AS
Admin Server
3. NM obtains startup properties for AS
2. NM determines domain directory, authenticates user
5. AS obtains domain configuration
Machine A

1. An authorized user issues the WLST offline command, `nmConnect` to connect to the Node Manager process on the machine that hosts the Administration Server, and issues a command to start the Administration Server. (If the Node Manager instance is the SSH version, the user can connect using the SSH client).

   The start command identifies the domain and server instance to start, and in the case of the Java Node Manager, provides the Node Manager username and password.

Note: If the user has previously connected to the Node Manager, a `boot.properties` file exists, and the user does not have to supply username and password.

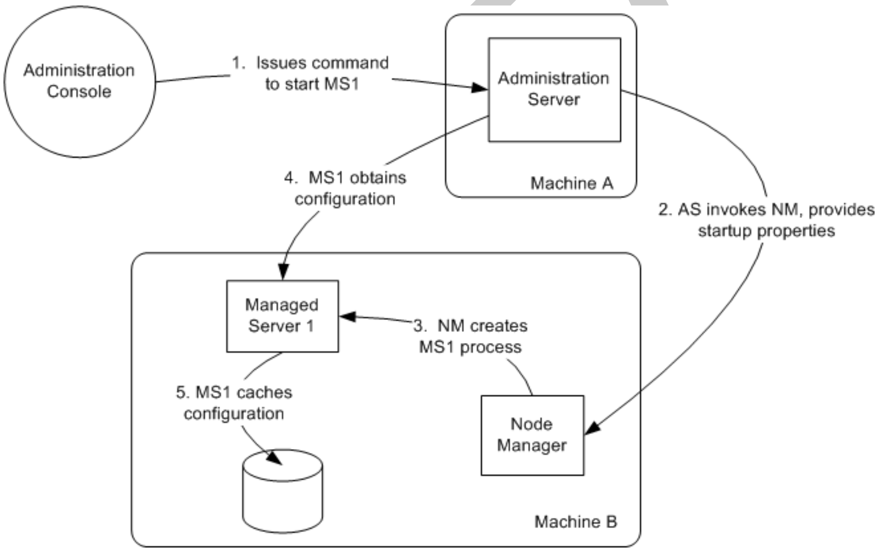
2. Node Manager looks up the domain directory in `nodemanager.domains`, and authenticates the user credentials using a local file that contains the encrypted username and password.
3. Node Manager creates the Administration Server process.
4. The Administration Server obtains the domain configuration from its `config` directory.

Starting a Managed Server With Node Manager

Figure 6-3 illustrates the process of starting a Managed Server with Node Manager.

Node Manager is running on Machine B, which hosts Managed Server 1. The Administration Server for the domain is running on Machine A.

Figure 6-3 Starting a Managed Server


Administration Console
1. Issues command to start MS1
Administration Server
Machine A
4. MS1 obtains configuration
2. AS invokes NM, provides startup properties
Managed Server 1
3. NM creates MS1 process
5. MS1 caches configuration
Node Manager
Machine B

1. From the Administration Console, the user issues a start command for Managed Server 1.

Note: A stand-alone client can also issue a start command for a Managed Server.

2. The Administration Server issues a start command for Managed Server 1 to the Node Manager on the Machine A, providing the remote start properties configured for Managed Server 1. For information about the arguments and how to specify them, see "Configure Remote Startup Arguments" on page 6-15.
3. Node Manager starts Managed Server 1.
4. Managed Server 1 contacts the Administration Server to check for updates to its configuration information.
5. If there are outstanding changes to the domain configuration, Managed Server 1 updates its local cache of configuration data.

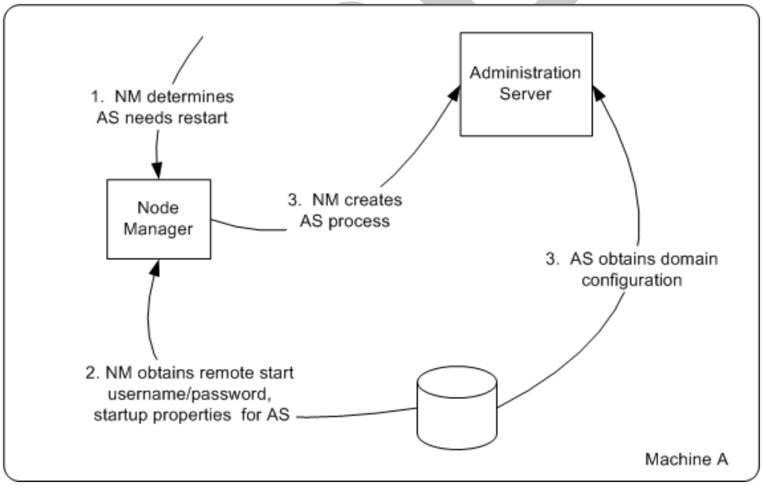
Restarting an Administration Server With Node Manager

Figure 6-4 illustrates the process of restarting a Administration Server with Node Manager.

Node Manager is running on the machine that hosts the Administration Server. The Administration Server, which was initially started with Node Manager, has exited. The Administration Server's `AutoRestart` attribute is set to `true`.

Note: If a server instance's `AutoRestart` attribute is set to `false`, Node Manager will not restart it.

Figure 6-4 Restarting Administration Server


1. NM determines AS needs restart
Administration Server
Node Manager
3. NM creates AS process
3. AS obtains domain configuration
2. NM obtains remote start username/password, startup properties for AS
Machine A

1. Node Manager determines from the Administration Server process exit code that it requires restart.

2. Node Manager obtains the username and password for starting the Administration Server from the `boot.properties` file, and the server startup properties from the `server/security/startup.properties` file. These server-specific files are located in the server directory for the Administration Server.

3. Node Manager starts the Administration Server.

4. The Administration Server reads its configuration data and starts up.

Restarting an Managed Server With Node Manager

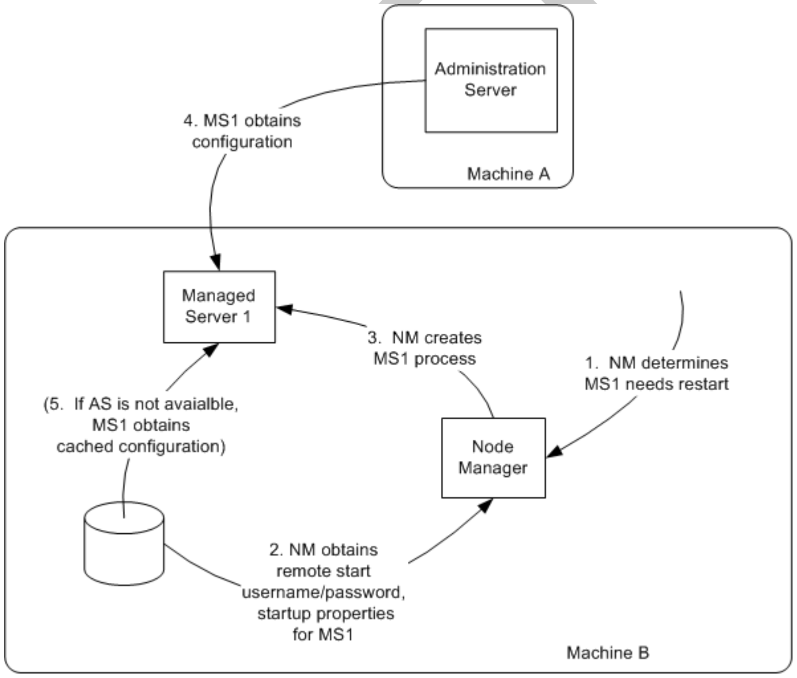
Figure 6-5 illustrates process of restarting a Managed Server with Node Manager.

Node Manager is running on Machine B, which hosts Managed Server 1. Managed Server 1, which was initially started with Node Manager, has exited. Managed Server 1's `AutoRestart` attribute is set to `true`.

Note: If a server instance's `AutoRestart` attribute is set to `false`, Node Manager will not restart it.

BETA

Figure 6-5 Restarting a Managed Server


Administration Server
Machine A
4. MS1 obtains configuration
Managed Server 1
3. NM creates MS1 process
1. NM determines MS1 needs restart
(5. If AS is not avaialble, MS1 obtains cached configuration)
Node Manager
2. NM obtains remote start username/password, startup properties for MS1
Machine B

1. Node Manager determines from Managed Server 1's last known state.
2. Node Manager obtains the username and password for starting Managed Server 1 from the `boot.properties` file, and the server startup properties from the `startup.properties` file. These server-specific files are located in the server directory for Managed Server 1.
3. Node Manager starts Managed Server 1

   Note: Node Manager waits `RestartDelaySeconds` after a server instances fails before attempting to restart it.
4. Managed Server 1 attempts to contact the Administration Server to check for updates to its configuration data. If it contacts the Administration Server and obtains updated configuration data, it updates its local cache of the `config` directory.
5. If Managed Server 1 fails to contact the Administration Server, and if Managed Server Independence mode (MSI) is enabled, Managed Server 1 uses its locally cached configuration data.

   Note: Managed Server Independence mode is enabled by default.

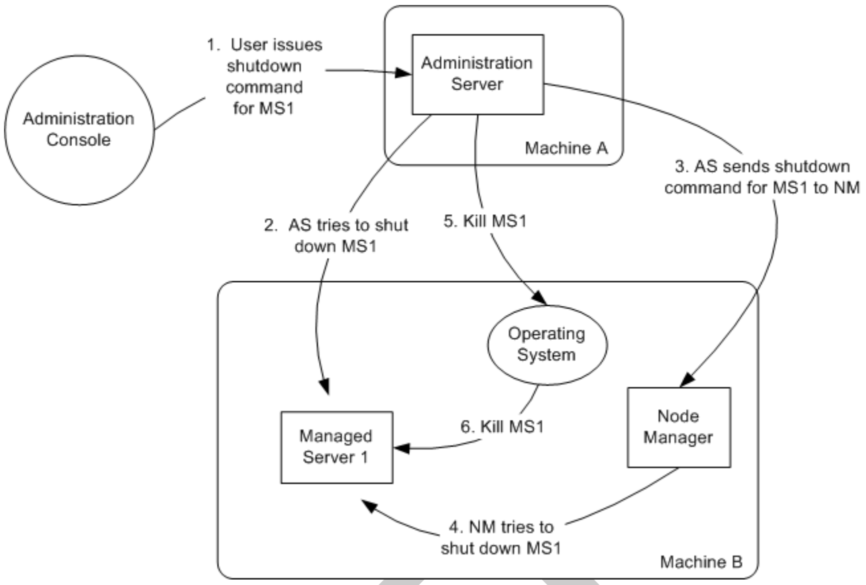
Stopping a Server Instance Under Node Manager Control

Figure 6-6 illustrates the communications involved in shutting down a Managed Server that is under Node Manager control. Depending on the state and availability of the Managed Server, Node Manager might need to try alternative strategies to successfully initiate the shutdown.

Node Manager is running on Machine B, which hosts Managed Server 1.

BETA

Figure 6-6 Shutting Down a Server Instance Under Node Manager Control


Administration Console
1. User issues shutdown command for MS1
Administration Server
Machine A
3. AS sends shutdown command for MS1 to NM
2. AS tries to shut down MS1
5. Kill MS1
Operating System
Node Manager
6. Kill MS1
Managed Server 1
4. NM tries to shut down MS1
Machine B

BETA

1. An authorized user issues a shutdown command for Manages Server 1 using the Administration Console.
2. The Administration Server issues a shutdown directly to Managed Server 1. If it successfully contacts Managed Servers 1, Managed Server1 performs the shutdown sequence described in "Graceful Shutdown" in *Managing Server Startup and Shutdown*.
3. If, in the previous step, the Administration Server failed to contact Managed Server 1, it issues shutdown command for Managed Server 1 to Node Manager on Machine B.
4. Node Manager issues a request to the operating system to kill Managed Server 1.
5. The operating system ends the Managed Server 1 process.

Setting Up Node Manager

Node Manager is ready-to-run after WebLogic Server installation if you run Node Manager and the Administration Server on the same machine, and use the demonstration SSL configuration. By default, the following behaviors are configured:

- You can start a Managed Server using Node Manager through the Administration Console.

- Node Manager monitors the Managed Servers that it has started.
- Automatic restart of Managed Servers is enabled. Node Manager restarts server instances that it killed or were killed by another method.

The following sections provide instructions for configuring Node Manager and the server instances it controls.

- "Configure Server Instances for Node Manager Control" on page 6-14—Follow the steps in this section to associate Node Manager with the server instances it controls, and configure desired restart behaviors. These configuration tasks apply, regardless of whether you are using the Java or the SSH version of Node Manager.
- Configure Node Manager—The process of configuring Node Manager varies, depending on which version you are using. As appropriate, follow the instructions in
  - "Configure Java Node Manager" on page 6-16, or

Configure Server Instances for Node Manager Control

This section describes how to configure the WebLogic server instances that Node Manager will control in your environment.

1. "Configure a Machine to Use Node Manager" on page 6-14
2. "Review nodemanager.domains" on page 6-14
3. "Configure Remote Startup Arguments" on page 6-15
4. "Ensure Administration Server Address is Defined" on page 6-16

Configure a Machine to Use Node Manager

A WebLogic Server Machine resource associates a particular system with the server instances it hosts, and specifies the connection attributes for the Node Manager process on that system.

Configure a machine definition for each machine that runs a Node Manager process using the Machines—>Configuration—>Node Manager page in the Administration Console. Enter the DNS name or IP address upon which Node Manager listens in the Listen Address box.

Review nodemanager.domains

The `nodemanager.domains` file specifies the domains that a Node Manager instance controls. This file must contain an entry specifying the domain directory for each domain, in this form:

BETA

```
<domain-name>=<domain-directory>
```

When a user issues a command for a domain, Node Manager looks up the domain directory from `nodemanager.domains`.

The client can only execute commands for the domains listed in `nodemanager.domains`.

If you created your domain with the Configuration Wizard, the `nodemanager.domains` file was created automatically. If necessary, you can manually edit `nodemanager.domains` to add a domain.

Configure Remote Startup Arguments

Specify the startup arguments that Node Manager will use to start a Managed Server in the Server—>Configuration—>Server Start page for the Managed Server. If you do not specify startup arguments for a Managed Server, Node Manager uses its own properties as defaults to start the Managed Server. For more information, see Table 6-1, "Node Manager Properties," on page 6-19. Although these defaults are sufficient to boot a Managed Server, to ensure a consistent and reliable boot process, configure startup arguments for each Managed Server.

If you will run Node Manager as a Windows Service, as described in "Starting Node Manager as a Service" on page 6-23, you must configure the following JVM property for each Managed Server that will be under Node Manager control:

- `-Xrs` for the Sun JVM, or
- `-Xnohup` for the Jrockit

If you do not set this option, Node Manager will not be able to restart a Managed Server after a system reboot, due to this sequence of events:

1. A reboot causes a running Managed Server to be killed before the Node Manager and Administration Server operating system services are shutdown.
2. During the interval between the Managed Server being killed, and the Node Manager service being shutdown, Node Manager continues to monitor the Managed Server, detects that it was killed, and attempts to restart it.
3. The operating system does not allow restart of the Managed Server because the machine is shutting down.
4. Node Manager marks the Managed Server as failed, and it will not start this server when the machine comes up again.

BETA

Starting a Managed Server with the `-Xrs` or `-Xnohup` option avoids this sequence of events by preventing the immediate shutdown of the Managed Server during machine shutdown.

Ensure Administration Server Address is Defined

Make sure that a Listen Address is defined for each Administration Server that will connect to the Node Manager process. If the Listen Address for an Administration Server is not defined, when Node Manager starts a Managed Server it will direct the Managed Server to contact localhost for its configuration information.

Set the Listen Address using the Servers—>Configuration—>General page in the Administration Console.

Configuring Node Manager to Use a Start Script to Start a Server

You can configure Node Manager to use a start script to start the server. This method allows you to use the script to set all the required properties for starting a server, as well as any other work you need performed at start up. To define a start script:

1. In the `nodemanager.properties` file, set the `StartScriptEnabled` property to `true`. If your start script is named `startWebLogic.sh` or `startWebLogic.cmd`, Node Manager uses one of those scripts as the default. If you want to specify the start script, see step 2.

2. In the `nodemanager.properties` file, set the `StartScriptName` property to the name of your script.

Configure Java Node Manager

Follow the instructions in the following section to configure the Java Node Manager:

1. Reconfigure Startup Service
2. Review nodemanager.properties
3. Configure SSL for Node Manager

Reconfigure Startup Service

The WebLogic Server installation process installs Node Manager as an operating system service or a Windows service on Windows systems. By default, the operating system service starts up Node Manager to listen on `localhost:5666`.

BETA

When you configure Node Manager to accept commands from remote systems, you must uninstall the default Node Manager service, then reinstall it to listen on a non-localhost Listen Address.

Depending on your platform, follow the instructions in "Reconfigure Startup Service for Windows Installations" or "Daemonizing Node Manager For UNIX Systems" below.

Reconfigure Startup Service for Windows Installations

The directory `WL_HOME\server\bin` (where *WL_HOME* is the top-level directory for the WebLogic Server installation) contains `uninstallNodeMgrSvc.cmd`, a script for uninstalling the Node Manager service, and `installNodeMgrSvc.cmd`, a script for installing Node Manager as a service.

1. Delete the service using `uninstallNodeMgrSvc.cmd`.
2. Edit `installNodeMgrSvc.cmd` to specify Node Manager's Listen Address and Listen Port.

   Make the same edits to `uninstallNodeMgrSvc.cmd` as you make to `installNodeMgrSvc.cmd`, so that you can successfully uninstall the service in the future, as desired.
3. Run `installNodeMgrSvc.cmd` to re-install Node Manager as a service, listening on the updated address and port.

Daemonizing Node Manager For UNIX Systems

WebLogic Server does not provide a command script for uninstalling and re-installing the Node Manager daemon process. Refer to your operating system documentation for instructions on uninstalling existing daemons, and setting up new ones.

1. Remove the Node Manager daemon process that the WebLogic Server installation process set up.
2. At the command line, or in a script, reinstall the Node Manager daemon. You may wish to view the contents of `installNodeMgrSvc.cmd` file before setting up the new daemon—although this command file is Windows-specific, it illustrates:
   – Key environment and local variables that must be defined.
   – Validation steps you might want to include in a script that installs Node Manager as a daemon.
   – Logic for setting default values for listen address and port.

BETA

3. To install Node Manager as a daemon, at the command line or in your script, you must, at a minimum:
   - Set `WL_HOME`
   - Set `NODEMGR_HOME`
   - Add the JDK and WebLogic directories to the system path.
   - Adds the JDK and WebLogic jars to the classpath.
   - Set `LD_LIBRARY_PATH`
   - Set `JAVA_VM`
   - Set `NODEMGR_HOST`
   - Set `NODEMGR_PORT`
   - Set `PROD_NAME`=BEA WebLogic Platform 9.0

Refer to your operating system documentation for operating system-specific settings that might be required.

Review nodemanager.properties

Node Manager properties define a variety of configuration settings for a Node Manager process. You can specify Node Manager properties on the command line or define them in the `nodemanager.properties` file, which is created in the directory where you start Node Manager the first time it starts up after installation of WebLogic Server. Values supplied on the command line override the values in `nodemanager.properties`.

`nodemanager.properties` is created in the directory where you start Node Manager the first time you start Node Manager after installation of WebLogic Server. Each time you start Node Manager, it looks for `nodemanager.properties` in the current directory, and creates the file if it does not exist in that directory. You cannot access the file until Node Manager has started up once.

Table 6-1 describes Node Manager properties.

In many environments, the SSL-related properties in `nodemanager.properties` may be the only Node Manager properties that you must explicitly define, as described in "Configure SSL for Node Manager" on page 6-22. However, `nodemanager.properties` also contains non-SSL properties in that you might need to specify, depending on your environment and preferences. For example:

BETA

- For a non-Windows installation, it might be appropriate to specify the `StartTemplate` and `NativeVersionEnabled` properties.
- If Node Manager runs on a multi-homed system, and you want to control which address and port it uses, define `ListenAddress` and `ListenPort`.

Table 6-1 Node Manager Properties

Node Manager Property	Description	Default
`CustomIdentityAlias` (DEPRECATED)	Specifies the alias when loading the private key into the keystore. This property is required when the `Keystores` property is set as `CustomIdentityandCustomTrust` or `CustomIdentityAndJavaStandardTrust`.	none
`CustomIdentityKeyStoreFileName` (DEPRECATED)	Specifies the file name of the Identity keystore (meaning the keystore that contains the private key for the Node Manager). This property is required when the `Keystores` property is set as `CustomIdentityandCustomTrust` or `CustomIdentityAndJavaStandardTrust`.	none
`CustomIdentityKeyStorePassPhrase` (DEPRECATED)	Specifies the password defined when creating the Identity keystore. This field is optional or required depending on the type of keystore. All keystores require the passphrase in order to write to the keystore. However, some keystores do not require the passphrase to read from the keystore. WebLogic Server only reads from the keystore, so whether or not you define this property depends on the requirements of the keystore.	none
`CustomIdentityKeyStoreType` (DEPRECATED)	Specifies the type of the Identity keystore. Generally, this is JKS. This property is optional	default keystore type from `java.security`

BETA

Node Manager Property	Description	Default
`CustomIdentityPrivateKeyPassPhrase` (DEPRECATED)	Specifies the password used to retrieve the private key for WebLogic Server from the Identity keystore. This property is required when the `Keystores` property is set as `CustomIdentityand CustomTrust` or `CustomIdentityAndJavaStandardTrust`.	none
`JavaHome` (DEPRECATED)	The Java home directory that Node Manager uses to start a Managed Servers on this machine, if the Managed Server does not have a Java home configured in its Remote Start page. If not specified in either place, Node Manager uses the Java home defined for the Node Manager process.	none
`KeyStores` (DEPRECATED)	Indicates the keystore configuration the Node Manager uses to find its identity (private key and digital certificate.) Possible values are: • `DemoIdentity`—Uses the demonstration identity keystore located in the `BEA_HOME\server\lib` directory. • `CustomIdentity`—Uses a keystore you create.	`DemoIdentityAndDemoTrust`
`ListenAddress` (DEPRECATED)	Any address upon which the machine running Node Manager can listen for connection requests. This argument deprecates `weblogic.nodemanager.listenAddress`.	null With this setting, Node Manager will listen on any IP address on the machine
`weblogic.nodemanager.listenAddress` (Deprecated)	The address on which Node Manager listens for connection requests. *Use* `ListenAddress` *in place of this deprecated argument.*	null With this setting, Node Manager will listen on any IP address on the machine

BETA

Node Manager Property	Description	Default
`ListenPort` (DEPRECATED)	The TCP port number on which Node Manager listens for connection requests. This argument deprecates `weblogic.nodemanager.listenPort`.	SSL: 5555 Plain: 5556 SSH: 22 RSH: 514
`weblogic.nodemanager.listenPort` (Deprecated)	The TCP port number on which Node Manager listens for connection requests. *Use `ListenPort` in place of this deprecated argument.*	5555
`NativeVersionEnabled`	For UNIX systems other than Solaris, HP-UX, or Linux, set this property to `false` to run Node Manager in non-native mode.	true
`WeblogicHome` (DEPRECATED)	Root directory of the WebLogic Server installation. This is used as the default value of `-Dweblogic.RootDirectory` for a Managed Server that does not have a root directory configured in its Remote Start page. If not specified in either place, Node Manager starts the Managed Server in the directory where Node Manager runs.	none
`LogFile`	Location of the Node Manager log file	`NM_HOME/nodemanager.log`
`LogLimit`	Maximum size of the Node Manager Log. When this limit is reached a new log file is started.	`true`
`LogCount`	Maximum number of log files to create when `LogLimit` is exceeded.	1
`LogAppend`	If set to `true`, then a new log file is not created when Node Manager restarts, the existing log is appended instead.	`true`
`LogToStderr`	If set to `true`, the log output is also sent to the standard error output.	`false`

BETA

Node Manager Property	Description	Default
`LogLevel`	Severity level of logging used for Node Manager log.May be one of the following levels: TRACE, DEBUG, INFO, WARNING, ERROR, NOTICE, CRITICAL, ALERT, EMERGENCY.	`WLLevel.ERROR`
`LogFormatter`	Name of formatter class to use for NM log messages.	`weblogic. nodemanager. server. LogFormatter`
`CrashRecoveryEnabled`	Enables system crash recovery.	`true`
`SecureListener`	If set to `true`, use the SSL listener, otherwise use the plain socket	`true`
`CipherSuite`	The name of the cipher suite to use with the SSL listener.	`TLS_RSA_EXPORT _WITH_RC4_40_M D5`
`StartScriptEnabled`	If true, use the start script specified by `StartScriptName` to start a server.	`false`
`StartScriptName`	The name of the start script, located in the domain directory	`startWebLogic. sh (UNIX)` or `startWebLogic. cmd (Windows)`
`DomainsFile`	The name of the `nodemanaer.domains` file	`NM_HOME/ nodemanager. domains`
`DomainsFileEnabled`	If set to `true`, use the file specified in `DomainsFile`. If `false`, assumes the domain of the current directory or of `WL_HOME`.	`true`

Configure SSL for Node Manager

The Java Node Manager communicates with Administration Servers and Managed Servers using one-way SSL.

BETA

The default WebLogic Server installation includes demonstration Identity and Trust keystores that allow you to use SSL out of the box. The keystores—`DemoIdentity.jks` and `DemoTrust.jks`—are installed in `*WLSHome*/server/lib`. For testing and development purposes, the keystore configuration is complete.

Configuring SSL for a production environment involves obtaining identity and trust for the Node Manager and each Administration and Managed Server with which the Node Manager will be communicating and then configuring the Node Manager, the Administration Server, and any Managed Servers with the proper identity and trust. In addition, the use of host name verification and the Administration port must be taken into consideration. To configure production SSL components, see "Configuring the SSL Protocol" in *Managing WebLogic Security*.

Starting and Stopping Node Manager

BETA

These sections describe methods of starting Node Manager, required environment variables, and command line arguments.

For prerequisite configuration steps see "Setting Up Node Manager" on page 6-13.

Starting Node Manager as a Service

The WebLogic Server installation process provides an option to install Node Manager as a Windows Service that it starts up automatically when the system boots. By default, Node Manager will listen on localhost. If you want Node Manager accept commands from remote systems, you must uninstall the default Node Manager service, then reinstall it to listen on a non-localhost Listen Address. For more information, see "Reconfigure Startup Service" on page 6-16.

Node Manager Environment Variables

Before starting Node Manager, you must set several environment variables. You can set the environment variables for a domain in a start script or on the command line. The sample start scripts provided with WebLogic Server—`startNodeManager.cmd` for Windows systems and `startNodeManager.sh` for UNIX systems—set the required variables, which are listed in the following table.

Table 6-2 Node Manager Environment Variables

Environment Variable	Description
`JAVA_HOME`	Root directory of JDK that you are using for Node Manager. For example: `set JAVA_HOME=c:\bea\jdk131` Node Manager has the same JDK version requirements as WebLogic Server.
`WL_HOME`	WebLogic Server installation directory. For example: `set WL_HOME=c:\bea\weblogic700`
`PATH`	Must include the WebLogic Server `bin` directory and path to your Java executable. For example: `set PATH=%WL_HOME%\server\bin;%JAVA_HOME%\bin;%PATH%`

`LD_LIBRARY_PATH` (UNIX only)	For HP UX and Solaris systems, must include the path to the native Node Manager libraries.
	Solaris example:
	`LD_LIBRARY_PATH:$WL_HOME/server/lib/solaris:$WL_HOME/server/lib/solaris/oci816_8`
	HP UX example:
	`SHLIB_PATH=$SHLIB_PATH:$WL_HOME/server/lib/hpux11:$WL_HOME/server/lib/hpux11/oci816_8`

`CLASSPATH`	You can set the Node Manager `CLASSPATH` either as an option on the `java` command line used to start Node Manager, or as an environment variable. Windows NT example: `set CLASSPATH=.;%WL_HOME%\server\lib\weblogic_sp.jar;%WL_HOME%\server\lib\weblogic.jar`

BETA

Starting the Java Node Manager with Commands or Scripts

Although running Node Manager as an operating system service is recommended, you can also start Node Manager manually at the command prompt or with a script. The environment variables Node Manager requires are described in "Node Manager Environment Variables" on page 6-23.

Sample start scripts for Node Manager are installed in the *`WL_HOME`*`\server\bin` directory, where *`WL_HOME`* is the top-level installation directory for WebLogic Server. Use `startNodeManager.cmd` on Windows systems and `startNodeManager.sh` on UNIX systems.

The scripts set the required environment variables and start Node Manager in *WL_HOME*/common/nodemanager. Node Manager uses this directory as a working directory for output and log files. To specify a different working directory, edit the start script with a text editor and set the value of the NODEMGR_HOME variable to the desired directory.

Edit the sample start script to make sure that the command qualifiers set the correct listen address and port number for your Node Manager process.

Command Syntax for Starting Node Manager

The syntax for starting Node Manager is:

```
java [java_option=value ...] -D[nodemanager_property=value]
-D[server_property=value] weblogic.NodeManager
```

where:

- *java_option* is a direct argument to the java executable, such as -ms or -mx.

  Note: If you did not set the CLASSPATH environment variable, use the -classpath option to identify required Node Manager classes.

- *nodemanager_property* is a Node Manager property. Instead of supplying Node Manager property values on the command line, you can edit the nodemanager.properties file, which is installed in the directory where you start Node Manager. For more information, see Table 6-1, “Node Manager Properties,” on page 6-19.

  Node Manager property values you supply on the command line override the values in nodemanager.properties.

- *server_property* is a server-level property that Node Manager accepts on the command line, including:
  - bea.home—the BEA home directory that server instances on the current machine use.
  - java.security.policy— path to the security policy file that server instances on the current machine use.

Notes: For UNIX systems:

If you run Node Manager on a UNIX operating system other than Solaris or HP UX, you cannot have any white space characters in any of the parameters that will be passed to the java command line when starting Node Manager. For example, this command fails due to the space character in the name “big iron”.

```
-Dweblogic.Name="big iron"
```

BETA

For UNIX systems other than Solaris, HP-UX, and Linux operating systems, you must disable the `weblogic.nodemanager.nativeVersionEnabled` option at the command line when starting Node Manager (or set the property in `nodemanager.properties`) to use the pure Java version. For more information. See "Review nodemanager.properties" on page 6-18.

Stopping Node Manager

To stop Node Manager, close the command shell in which it is running.

Using the SSH Node Manager Command Shell

To use the SSH Node Manager Command Shell, start the Administration Server using the following command line option:

```
-Dweblogic.nodemanager.ShellCommand='ssh -o PasswordAuthentication=no %H
wlscontrol.sh -d %D -r %R -s %S -v %C'
```

The `weblogic.nodemanager.ShellCommand` attribute specifies the command template to use to communicate with a remote SSH Node Manager and execute Node Manager functions for server instances under its control.

The template assumes that `wlscontrol.sh` is in the default search path on the remote machine hosting Node Manager.

The `ShellCommand` syntax is:

```
ssh -o PasswordAuthentication=no %H wlscontrol.sh -d %D -r %R -s %S -v %C'
```

where the parameter values are provided for each of the parameters, as defined in Table 6-3.

For example, if you type this command,

```
ssh -o PasswordAuthentication=no wlscontrol.sh myserver start
```

The listen address and port of the SSH server default to the listen address and port used by Node Manager on the remote machine, and the domain name and directory, they are assumed to be the root directory specified for the target server instance, `myserver`.

This command:

```
ssh -o PasswordAuthentication=no 172.11.111.11 wlscontrol.sh -d
ProductionDomain -r ProductionDomain -s ServerA -v START'
```

issues a `START` command to the server instance named `ServerA`, in the domain called `ProductionDomain`, located in the `domains/ProductionDomain` directory.

BETA

Table 6-3 Shell Command

Parameter	Description	Default
%H	Host name of SSH server	`NodeManagerMBean.ListenAddress`
%P	Port number of SSH server	`NodeManagerMBean.ListenAddress` 22
%S	WebLogic server name	none
%D	WebLogic domain name	`ServerStartMBean.RootDirectory`
%R	Domain directory (server root)	`ServerStartMBean.RootDirectory`
%C	Node manager script command • `START`—Start server • `KILL`—Kill server • `STAT`—Get server status • `GETLOG`—Retrieve server output log. • `GETNMLOG`—Retrieve regular server log, which contains `STDOUT` and `STDERR` output. • `VERSION`—Return Node Manager version.	none

BETA

The ssh command must include the string:

```
-o PasswordAuthentication=no
```

This string passes the ssh `PasswordAuthentication` option. A value of `yes` causes the client will hang when it tries to read from the console.

Node Manager Log Files and Error Messages

The following sections describe how to diagnose and correct Node Manager problems. Use the Node Manager log files to help troubleshoot problems in starting or stopping individual Managed Servers. Use the steps in "Correcting Common Problems" on page 6-29 to solve problems in Node Manager configuration and setup.

Node Manager Log File

Node Manager creates a log file located in `NM_HOME/nodemanager.log`. Log rotation is disabled by default, but can be enabled by setting `LogCount` in `nodemanager.properties`. Also, by default, log output will be appended to the current `nodemanager.log`. These and other logging-related properties are based on the logging properties specified in the Java 2 logging APIs in `java.util.logging`.

You can view the Node Manager log file by:

- Selecting Machines—>Monitoring—>Node Manager Log page in the Administration Console
- Using the WLST `nmLog` command

Node Manager Log File For Each Server Instance

For each server instance that it controls, Node Manager maintains a log file that contains `stdout` and `stderr` messages generated by the server instance. If the remote start debug property is enabled as a remote start property for the server instance, or if the NodeManager debug property is enabled, Node Manager will include additional debut information in the server output log information.

Note: You cannot limit the size of the log files Node Manager creates. Logging to `stdout` is disabled by default .

Node Manager creates the server output log for a server instance in the server instance's `logs` directory, with the name:

`server-name``.out`

where *`server-name`* is the name of the server instance.

- Using the WLST `nmServerLog` command

There is no limit to the number of server output logs that Node Manager can create.

Server Log Files

A server instance under Node Manager control has its own log file, in addition to the log file created by Node Manager. View the regular log file for a server instance by selecting Diagnostics—>Log Files, selecting the server log file, and clicking View.

BETA

Correcting Common Problems

The table below describes common Node Manager problems and their solutions

Table 6-4 Troubleshooting Node Manager.

Symptom	Explanation
Error message: `Could not start server 'MyServer' via Node Manager - reason: 'Target machine configuration not found'.`	You have not assigned the Managed Server to a machine. Follow the steps in "Configure a Machine to Use Node Manager" on page 6-14.
Error message: `<SecureSocketListener: Could not setup context and create a secure socket on 172.17.13.26:7001>`	The Node Manager process may not be running on the designated machine. See "Starting and Stopping Node Manager" on page 6-23.

BETA

Symptom	Explanation
I configured self-health monitoring attributes for a server, but Node Manager doesn't automatically restart the server.	To automatically reboot a server, you must configure the server's automatic restart attributes as well as the health monitoring attributes. See "Starting and Stopping Node Manager" on page 6-23. In addition, you must start Managed Servers using Node Manager. You cannot automatically reboot servers that were started outside of the Node Manager process (for example, servers started directly at the command line).

Applications on the Managed Server are using the wrong directory for lookups.	Applications deployed to WebLogic Server should never make assumptions about the current working directory. File lookups should generally take place relative to the Root Directory obtained with the `ServerMBean.getRootDirectory()` method (this defaults to the "." directory). For example, to perform a file lookup, use code similar to
	
	`String rootDir = ServerMBean.getRootDirectory();`
	` //application root directory`
	`File f = new File(rootDir + File.separator +`
	` "foo.in");`
	
	rather than simply:
	
	`File f = new File("foo.in");`
	
	If an application is deployed to a server that is started using Node Manager, use the following method calls instead:
	
	`String rootDir //application root directory`
	`if ((rootDir = ServerMBean.getRootDirectory()) ==`
	` null) rootDir =`
	` ServerStartMBean.getRootDirectory();`
	`File f = new File(rootDir + File.separator +`
	` "foo.in");`
	
	The `ServerStartMBean.getRootDirectory()` method obtains the Root Directory value that you specified when configuring the server for startup using Node Manager. (This corresponds to the `Root Directory` attribute specified the Configuration—>Remote Start page of the Administration Console.)

BETA

Node Manager and Managed Server States

Node Manager defines its own, internal Managed Server states for use when restarting a server. If Node Manager is configured to restart Managed Servers, you may observe these states in the Administration Console during the restart process.

- `FAILED_RESTARTING`—Indicates that Node Manager is currently restarting a failed Managed Server.

BETA

BETA

Using the WebLogic Persistent Store

The following sections explain how to configure and monitor the persistent store, which provides a built-in, high-performance storage solution for WebLogic Server subsystems and services that require persistence.

BETA

Overview of the Persistent Store

The persistent store provides a built-in, high-performance storage solution for WebLogic Server subsystems and services that require persistence. For example, it can store persistent JMS messages or temporarily store messages sent using the Store-and-Forward feature. The persistent store supports persistence to a file-based store or to a JDBC-enabled database.

Table 7-1 defines the WebLogic services and subsytems that can create connections to the persistent store. Each subsystem that uses the persistent store specifies a unique Connection ID that identifies that subsystem.

Table 7-1 Persistent Store Users

Subsystem/Service	What It Stores	More Information
JMS Servers	Persistent messages and durable subscribers	*Configuring and Managing WebLogic JMS*
Store-and-Forward (SAF) Service Agents	Storing messages for a sending SAF agent for retransmission to a receiving SAF agent	*Configuring and Managing WebLogic Store-and-Forward Service*
Transaction Log (TLOG)	Information about committed transactions coordinated by the server that may not have been completed	"Managing Transactions" in *Programming WebLogic JTA*
Path Service	TBD	TBD
Web Services	TBD	TBD
EJB Timer Services	TBD	TBD

BETA

Features of the Persistent Store

The key features of the persistent store include:

- Default file store for each server instance that requires no configuration
- One store is shareable by multiple subsystems, as long as they are all targeted to the same server instance or migratable target
- High-performance throughput and transactional support

- Modifiable parameters that let you create customized file stores and JDBC stores
- Monitoring capabilities for persistent store statistics and open store connections

High-Performance Throughput and Transactional Support

Throughput is the main performance goal of the persistent store. Multiple subsystems can share the same persistent store, as long as they are all targeted to the same server instance or migratable target.

This is a performance advantage because the persistent store is treated as a single resource by the transaction manager for a particular transaction, even if the transaction involves multiple services that use the same store. For example, if JMS and EJB timers share a store, and a JMS message and an EJB timer are created in a single transaction, the transaction will be one-phase and incur a single resource write, rather than two-phase, which incurs four resource writes (two on each resource), plus a transaction entry write (on the transaction log).

Both the file store and the JDBC store can survive a process crash or hardware power failure without losing any committed updates. Uncommitted updates may be retained or lost, but in no case will a transaction be left partially complete after a crash.

Comparing File Stores and JDBC Stores

The following are some similarities and differences between file stores and JDBC stores:

- The default persistent store can only be a file store.
- A JDBC store cannot be used as a default persistent store.
- The transaction log (TLOG) can only be stored in a file store.
- Both have the same transaction semantics and guarantees. As with JDBC store writes, file store writes are guaranteed to be persisted to disk and are not simply left in an intermediate (that is, unsafe) cache.
- Both have the same application interface (no difference in application code).
- All things being equal, file stores generally offer better throughput than a JDBC store.

  Note: If a database is running on high-end hardware with very fast disks, and WebLogic Server is running on slower hardware or with slower disks, then you may get better performance from the JDBC store.

BETA

- File stores are generally easier to configure and administer, and do not require that WebLogic subsytems depend on any external component.
- File stores generate no network traffic; whereas, JDBC stores will generate network traffic if the database is on a different machine from WebLogic Server.
- JDBC stores may make it easier to handle failure recovery since the JDBC interface can access the database from any machine on the same network. With the file store, the disk must be shared or migrated.

Using the Default Persistent Store

Each server instance, including the administration server, has a default persistent store that requires no configuration. The default store is a file-based store that maintains its data in a group of files in a server instance's `data\store\default` directory. In fact, a directory for the default store is automatically created if one does not already exist. This default store is available to subsystems that do not require explicit selection of a particular store and function best by using the system's default storage mechanism. For example, a JMS Server with no persistent store configured will use the default store for its Managed Server and will support persistent messaging.

The default store can be configured by directly manipulating the `DefaultFileStoreMBean` parameters. If this MBean is not defined in the domain's configuration file, then the configuration subsystem ensures that the `DefaultFileStoreMBean` always exists with the default values.

In addition to using the default file store, you can also configure a custom file store or JDBC store to suit your specific needs, as explained in "Using Custom File Stores and JDBC Stores" on page 7-6. One exception, however, is the transaction log (TLOG), which always uses the default store. This is because the transaction log must always be available early in the server boot process.

BETA

Default Store Location

The default store maintains its data in a `data\store\default` directory inside the *`servername`* subdirectory of a domain's root directory

For example, if no directory name is specified for the default file store, it defaults to:

```
bea_home\user_projects\domains\domainname\servers\servername\data\store
\default
```

where *`domainname`* is the root directory of your domain, typically `c:\bea\user_projects\domains\`*`domainname`*, which is parallel to the directory in which WebLogic Server program files are stored, typically `c:\bea\weblogic90`.

You can, however, specify another location for the default store using any of the methods described in "Methods of Creating a Persistent Store" on page 7-6.

Example of a Default File Store

Here's an example of how a default file store may look in a domain's configuration file, with the directory location and Synchronous Write Policy of the default file store overridden:

```
<server>
<name>myserver</name>
<default-file-store>
<directory>C:/store</directory>
<synchronous-write-policy>Disabled</synchronous-write-policy>
</default-file-store>
</server>
```

Default Store Migration

For high availability, the default persistent store can be migrated as part of a "whole server" migration, as discussed in the "Server Migration" section of *Using WebLogic Server Clusters*.

In addition, singleton services that use a migratable target, such as JMS and the JTA transaction recovery system, which run only on one server in the cluster at any given time, can also use the default store, as long as the default store is accessible by the target server instance.

A default store can also be migrated as part of the migration of a "singleton service" that uses a migratable target, such as JMS and the JTA transaction recovery systems, which run only on one server in the cluster at any given time. However, this means that a default store must be configured on a shared disk that is available to the migratable target server. Therefore, applications that need access to persistent stores that reside on remote machines after the migration of a JMS server or JTA transaction log must implement an alternative solution, as follows:

- Default store or custom file store — Configure a default store or custom file store and implement a hardware solution, such as a dual-ported SCSI disk or Storage Area Network (SAN) to make your file store available from remote machines.

BETA

- JDBC Store — Configure a JDBC store and use JDBC to access this store, which can be on yet another server. Applications can then take advantage of any high-availability or failover solutions offered by your database vendor.

For more information on migrating singleton services, see "Migration for Singleton Services" in *Using WebLogic Server Clusters*.

Using Custom File Stores and JDBC Stores

In addition to using the default file store, you can also configure a file store or JDBC store to suit your specific needs. A custom file store, like the default file store, maintains its data in a group of files in a directory. However, you may want to create a custom file store so that the file store's data is persisted to a particular storage device. When configuring a file store directory, the directory must be accessible to the server instance on which the file store is located.

A JDBC store can be configured when a relational database is used for storage. A JDBC store enables you to store persistent messages in a standard JDBC-capable database, which is accessed through a designated JDBC data source. The data is stored in the JDBC store's database table, which has a logical name of `WLStore`. It is up to the database administrator to configure the database for high availability and performance.

For more information about configuring a persistent store, see "When to Use a Custom Persistent Store" on page 7-6.

When to Use a Custom Persistent Store

WebLogic Server provides configuration options for creating a custom file store or JDBC store. For example, you may want to:

- Place a file store's files on a particular device.
- Use a JDBC store rather than a file store for a particular server instance.
- Allow all physical stores in a cluster to share the same logical name.
- Logically separate different services to use different files or tables. (This may simplify administration and maintenance at the expense of reduced performance.)

Methods of Creating a Persistent Store

A customized persistent store can be configured in the following ways:

BETA

- Use the Administration Console to configure a custom file store or JDBC store, as described in `http://bt04/stage/wls/docs90/ConsoleHelp/index.html`
- Directly edit the configuration file (`config.xml`) of the server instance that is hosting the default persistent store.
- Use the WebLogic Java Management Extensions (JMX) to create persistent stores. JMX is the J2EE solution for monitoring and managing resources on a network. For more information see, "Programming WebLogic Management Services with JMX".
- Use the WebLogic Scripting Tool (WLST) to create persistent stores. WLST is a command-line scripting interface that you use to interact with and configure WebLogic Server instances and domains. For more information, see "WebLogic Scripting Tool".
- Use the WebLogic Configuration Wizard to change the options of the default persistent store. For detailed information on how to use the Configuration Wizard to configure a persistent store, see "Creating a New WebLogic Domain".

Creating a Custom (User-Defined) File Store

The following sections provide an example of a custom file store and configuration guidelines for choosing a synchronous write policy.

To create a custom file store, you can directly modify the default `FileStoreMBean` parameters. For instructions on using the Administration Console to create a custom file store, see [CONSOLE HELP LINK].

BETA

Main Steps for Configuring a Custom File Store

The main steps for creating a custom file store are as follows:

1. Create a directory where the file store's data will be persisted.
2. Create a custom file store and specify the directory location that you created.
3. Associate the custom file store with the subsystem(s) that will be using it, such as:
   – For JMS servers, select the custom file store on the General Configuration page.
   – For Store-and-Forward agents, select the custom file store on the General Configuration page.

Example of a Custom File Store

Here's an example of how a custom file store may look in a domain's configuration file with its files kept in a `/disk1/jmslog` directory.

```
<file-store>
<name>SampleFileStore</name>
<target>myserver</target>
<directory>/disk1/jmslog</directory>
</file-store>
```

Table 7-2 briefly describes the file store configuration parameters that can be modified.

Table 7-2 File Store Configuration Options

Option	Required	What it does. . .
Name	Yes	The name of the file store, which must be unique across all stores in the domain.
Targets	Yes	The server instance where a file store is targeted. Multiple subsystems can share the same store, as long as they are all targeted to the same server instance or migratable target.
Directory	Yes	The path name to the directory on the file system where the file store is kept.
Logical Name	No	Optionally used with subsystems, like EJBs, when deploying a module to an entire cluster, but also require a different physical store on each server instance in the cluster. In such a configuration, each physical store would have its own name, but all the persistent stores would share the same logical name. [VERIFY WITH EJB DOCS AND ADD LINK TO THEM]
Synchronous Write Policy	No	Optionally defines how hard a file store will try to flush records to the disk. Values are: Direct-Write (default), Cache-Flush, and Disabled. For more information, see "Guidelines for Configuring a Synchronous Write Policy" on page 7-9.

For step-by-step instructions on configuring a custom file store, see [LINK TO CONSOLE HELP]

BETA

Guidelines for Configuring a Synchronous Write Policy

With the default Synchronous Write Policy of Direct-Write, file store writes are written directly to disk for Solaris and Windows operating systems. On Windows systems, this option generally performs faster than the Cache-Flush option.

Changing the default policy to Disabled generally improves file store performance, often quite dramatically, but at the expense of possibly losing sent messages or generating duplicate received messages (even if messages are transactional) in the event of an operating system crash or a hardware failure. This is because transactions are complete as soon as their writes are cached in memory, instead of waiting for the writes to successfully reach the disk. Simply shutting down an operating system does not generate these failures, as an OS flushes all outstanding writes during a normal shutdown. Instead, these failures can be emulated by abruptly shutting the power off to a busy server.

Creating a JDBC Store

The following sections provide an example of a JDBC store, and information about creating a database table for a JDBC store, either using existing DDL or It also includes instruction on enabling Oracle blob record columns in a DDL file.

To create a JDBC store, you can directly modify the default `JDBCStoreMBean` parameters. For instructions on using the Administration Console to create a JDBC store, see [CONSOLE HELP].

For configuration guidelines on using prefixes with JDBC stores and recommended JDBC data source settings, see "Guidelines for Configuring a JDBC Store" on page 7-15.

Main Steps for Configuring a JDBC Store

The main steps for creating a JDBC store are as follows:

1. Create a JDBC data source or multi data source to interface with the JDBC store.
2. Create a JDBC store and associate it with the JDBC data source or multi data source.
3. It is highly recommended that you configure the Prefix option to a unique value for each configured JDBC store table.
4. Associate the JDBC store with the subsystem(s) that will be using it, such as:
   – For JMS servers, select the JDBC store on the General Configuration page.

BETA

– For Store-and-Forward agents, select the JDBC store on the General Configuration page.

Example of a JDBC Store

Here's an example of how a JDBC store may look in the configuration file, using the JDBC data source `MyDataSource`, and with a logical name specified:

```
<jdbc-store>
<name>SampleJDBCStore</name>
<target>yourserver</target>
<data-source>MyDataSource</data-source>
<logical-name>Baz</logical-name>
</jdbc-store>
```

Table 7-3 describes the JDBC store configuration parameters that can be modified.

Table 7-3 JDBC Store Configuration Option

Option	Required	What it does. . .
Name	Yes	The name of the JDBC store, which must be unique across all stores in the domain.
Targets	Yes	The server instance where a JDBC store is targeted. Multiple subsystems can share the same store, as long as they are all targeted to the same server instance or migratable target.
Data Source	Yes	The JDBC data source or multi data source used by this JDBC store to access the store's database table (`WLStore`). The specified JDBC data source must use a non-XA JDBC driver; data sources for XA JDBC drivers are not supported. This data source or multi data source must be targeted to the same server instance as the JDBC store.
Prefix Name	No	The prefix for the JDBC store's table is generally entered in the following format: `[[[catalog.]schema.]prefix]` It is required to set this option to a unique value for each configured JDBC store. When no prefix is specified, the JDBC store table name is simply `WLStore` and the database implicitly determines the schema according the current user of the JDBC connection. Also, two JDBC stores cannot share the same database table. For more information, see "Using Prefixes with a JDBC Store" on page 7-15.

BETA

Table 7-3 JDBC Store Configuration Option

Option	Required	What it does. . .
Logical Name	No	Optionally used with WebLogic Server subsystems, like EJBs, when deploying a module to an entire cluster, but also require a different physical store on each server instance in the cluster. In such a configuration, each physical store would have its own name, but all the persistent stores would share the same logical name.
DDL File Name	No	Optionally used with supported DDL (data definition language) files to create the JDBC store's database table (`WLStore`). This option is ignored when the JDBC store's database table already exists. For more information, see "Automatically Creating a JDBC Store Table Using Default and Custom DDL Files" on page 7-12.

For step-by-step instructions on configuring a JDBC store, see [LINK TO CONSOLE HELP]

Supported JDBC Drivers

When using a JDBC store, the backing database can be any database that is accessible through a JDBC driver. WebLogic Server detects some drivers for supported databases.

For each of these databases, there are corresponding DDL (data definition language) files within the *WL_HOME*`\server\lib\weblogic.jar` file, in the `weblogic/store/io/jdbc/ddl` directory, where *`WL_HOME`* is the top-level directory of your WebLogic Server installation.

Table 7-4 Supported JDBC Drivers and Corresponding DDL Files

Database	DDL Files
Cloudscape	`cloudscape.ddl`
IBM DB2	`db2.ddl` `db2v6.ddl`
Informix	`informix.ddl`
Microsoft SQL (MSSQL) Server	`msssql.ddl`
HP NonStop SQL	`nssql.ddl`
MySQL	`mysql.ddl`

BETA

Table 7-4 Supported JDBC Drivers and Corresponding DDL Files

Database	DDL Files
Oracle	`oracle.ddl` `oracle_blob.ddl`
Pointbase	`pointbase.ddl`
Sybase	`sysbase.ddl`
Times Ten	`timesten.ddl`

The DDL files are actually text files containing the SQL commands (terminated by semicolons) that create the JDBC store's database table (`WLStore`). Therefore, if you are using a database that is not included in this list, you can copy and edit any one of the existing DDL files to suit your specific database, as described in "Creating a JDBC Store Table Using a Custom DDL File" on page 7-13.

BETA

Automatically Creating a JDBC Store Table Using Default and Custom DDL Files

The JDBC Store Configuration page provides an optional DDL File Name option, through which you can access a pre-configured DDL file that is used to create the JDBC store's database table (`WLStore`). This option is ignored when the JDBC store's database table already exists. It is mainly used to specify a custom DDL file created for an unsupported database, or when upgrading a 8.1 or earlier JMS JDBC store tables to a release 9.0 JDBC Store table.

If the DDL File Name option is *not* configured and the JDBC store detects that its database table does not already exist, the JDBC store automatically creates the database table by executing a pre-configured DDL file that is specific to the database vendor (see Supported JDBC Drivers).

If the DDL File Name option is configured and the JDBC store detects that its database table does not already exist, the JDBC store searches for the indicated file in the file path first, then, if not found, searches for the file in the `CLASSPATH`. Once found, the SQL within the file is executed to create the JDBC store's database table. If the configured file is not found and the database table doesn't already exist, the JDBC store will fail to boot.

Creating a JDBC Store Table Using a Custom DDL File

To use a different database from those listed in "Supported JDBC Drivers" on page 7-11, you can copy and edit any one of the existing DDL template files to suit your specific database.

1. Use the JAR utility supplied with the JDK to extract the DDL files to the `/weblogic/store/io/jdbc/ddl` directory using the following command:

   ```
   jar xf weblogic.jar /weblogic/store/io/jdbc/ddl
   ```

   Note: If you omit the `weblogic/store/io/jdbc/ddl` parameter, the entire jar file is extracted.

2. Edit the DDL file for your database. An SQL command can span several lines and is terminated with a semicolon (;). Lines beginning with pound signs (#) are comments.

3. Save your changes and rename the new DDL appropriately (for example, `mydatabase.ddl`)

4. Create a JDBC store, as explained in "Creating a JDBC Store".
   [TBD To Console Help]

5. Use the DDL File Name option on the General Configuration page to specify your custom DDL file (for example, `/mydatabase.ddl`

   Note: On Windows systems, for full path names always include the drive letter.

Enabling Oracle BLOB Record Columns

For Oracle databases, you can use the `oracle_blob.ddl` file to create a JDBC store table with a BLOB record column type rather than the default LONG RAW record column type. The `oracle_blob.ddl` file is pre-configured and supplied in the WebLogic `CLASSPATH`, as described in "Supported JDBC Drivers" on page 7-11.

To use the Oracle BLOB DDL with a JDBC store:

1. Shut down the server instance that uses the JDBC store.

2. Delete the current JDBC table, as explained in "Managing JDBC Store Tables" on page 7-14.

3. Reboot the server instance.

4. Create a new JDBC store, as explained in "Creating a JDBC Store"
   [TBD To Console Help].

BETA

5. In the DDL File Name field on the General Configuration page, enter the location of the `oracle_blob.ddl` file, as follows: `/oracle_blob.ddl`
6. Click Finish to create the

If you need to preserve data already in a Oracle LONG RAW column, but still want to switch the column to BLOB, *do not* use this method. Instead, consult the Oracle documentation for the SQL ALTER TABLE command.

Managing JDBC Store Tables

The JDBC `utils.Schema` utility allows you to regenerate a new JDBC store database table (`WLStore`) by deleting the existing version. Running this utility is usually not necessary, since WebLogic Server automatically creates this table for you. However, if your existing JDBC store database table somehow becomes corrupted, you can delete it using the `utils.Schema` utility.

The `utils.Schema` utility is a Java program that takes command-line arguments to specify the following:

- JDBC driver
- Database connection information
- Name of a file containing the SQL Data Definition Language (DDL) commands that create the database table

BETA

Using the utils.Schema Utility to Delete a JDBC Store Table

Enter the `utils.Schema` command, as follows:

```
$ java utils.Schema url JDBC_driver [options] DDL_file
```

Note: To execute `utils.Schema`, your `CLASSPATH` must contain the `weblogic.jar` file.

Table 7-5 lists the `utils.Schema` command-line arguments.

Table 7-5 Command-line arguments for utils.Schema

Argument	Description
url	Database connection URL. This value must be a colon-separated URL as defined by the JDBC specification.
JDBC_driver	Full package name of the JDBC Driver class.

Table 7-5 Command-line arguments for utils.Schema

Argument	Description

options	Optional command options.
	If required by the database, you can specify:
	• The username and password as follows:
	`-u <username> -p <password>`
	• The Domain Name Server (DNS) name of the JDBC database server as follows:
	`-s <dbserver>`
	You can also specify the `-verbose` option, which causes `utils.Schema` to echo SQL commands as they are executed.

DDL_file	The full pathname of the DDL text file containing the SQL commands that you want to execute. For more information, see "Supported JDBC Drivers" on page 7-11.

For example, the following command recreates the JDBC tables in an Oracle server named `DEMO`, with the username `user1` and password `foobar`:

```
$ java utils.Schema
jdbc:weblogic:oracle:DEMO \
weblogic.jdbc.oci.Driver -u user1 -p foobar -verbose \
weblogic/store/io/jdbc/ddl/oracle.ddl
```

BETA

Guidelines for Configuring a JDBC Store

The following sections provide guidelines for using JDBC store prefixes, recommended WebLogic JDBC data source settings for JDBC stores, and handling JMS transactions with JDBC stores.

Using Prefixes with a JDBC Store

The JDBC store database contains a database table, named `WLStore`, that is generated automatically and is used internally by WebLogic Server. The JDBC store provides an optional Prefix Name parameter, which can be used to provide more precise access to the database table.

It is always a best practice to configure a prefix for the JDBC `WLStore` table name, especially when:

- The database requires fully-qualified names. (You should verify this with your database administrator.)
- There is more than one JDBC store instance sharing a database, since no two JDBC stores can share the same table.
- There are many tables in the database. Setting the prefix reduces the number of tables the JDBC store must search through to find its table during boot.

JDBC Store Table Rules

To avoid potential data loss, follow these rules:

- Each JDBC store table name must be unique.
- If multiple JDBC stores share a table, the behavior is undefined and data loss is likely.
- There is no procedure for combining two database tables into a single table.

Prefix Name Format Guidelines

For most databases, the Prefix Name option for the JDBC store's backing database table should be set in the following format for each configured JDBC store, which will result in a valid table name when prepended to the JDBC store table name:

```
[[[catalog.]schema.]prefix]
```

Note that each period in the `[[[catalog.]schema.]prefix]` format is significant. Generally, *`catalog`* identifies the set of system tables being referenced by the DBMS, and *`schema`* generally corresponds to ID of the table owner (*`username`*). When no prefix is specified, the JDBC store table name is simply `WLStore` and the database implicitly determines the schema according the current user of the JDBC connection.

For example, in a production database, the database administrator could maintain a unique table for the Sales department, as follows:

```
[[[Production.]JMSAdmin.]Sales]
```

The resulting table will be created in the Production catalog, under the JMSAdmin schema, and will be named `SalesWLStore`.

For some DBMS vendors, such as Oracle, there is no catalog to set or choose, so the format simplifies to `[[schema.]prefix]`. For more information, refer to your DBMS documentation

BETA

for instructions on fully-qualified table names, but note that the syntax specified by the DBMS may differ from the format required for this option.

Caution: If the Prefix Name setting is changed, but the `WLStore` database table already exists in the database, take care to preserve existing table data. In this case, the existing database table must be renamed by a database administrator to match the new configured table name.

Recommended JDBC Data Source Settings for JDBC Stores

The following settings are recommended when you use a JDBC data source or multi data source for JDBC stores.

Automatic Reconnection to Failed Databases

WebLogic Server provides robust JDBC data sources that can automatically reconnect to failed databases after they come back online, without requiring you to restart WebLogic Server. To take advantage of this capability, and make your use of JDBC stores more robust, configure the following options on the JDBC data source associated with the JDBC store:

```
TestConnectionsOnReserve="true"
TestTableName="SYSTABLES"
ConnectionCreationRetryFrequencySeconds="600"
```

For more information about JDBC default Test Table Names, see "Connection Testing Options" in the *Configuring and Managing WebLogic JDBC*. For more information about setting the number of database reconnection attempts, see the "Enabling Connection Creation Retries" section in *Programming WebLogic JDBC*.

Required Setting for WebLogic Type 4 JDBC DB2 Drivers

For data sources used as a JDBC store that use the WebLogic Type 4 JDBC driver for DB2, the `BatchPerformanceWorkaround` property must be set to "true" due to internal JMS batching requirements.

For more information, see the "Performance Workaround for Batch Inserts and Updates" section in the *WebLogic Type 4 JDBC Drivers* documentation.

Handling JMS Transactions with JDBC Stores

You cannot configure a JDBC store to use a JDBC data source that is configured to support global transactions. The JDBC store must use a JDBC data source that uses a non-TxDataSource with a

BETA

non-`XAResource` driver (you cannot use an XA driver or a JTS driver, or implement the Emulate Two-Phase Commit for non-XA Driver option). The JDBC store does the XA support above the JDBC driver.

[NEEDS TO BE REVIEWED BY RAHUL/DAVE AFTER THE JDBC DOCS HARDEN]

This is because the WebLogic store is its own resource manager. That is, the store itself implements the `XAResource` and handles the transactions without depending on the database (even when the messages are stored in the database). This means that whenever you are using a JDBC store and a database (even if it is the same database where the JMS messages are stored), then it is two-phase commit transaction. For more information about using JMS transactions with a JDBC store, see "Using Transactions with WebLogic JMS" in *Programming WebLogic JMS*.

From a performance perspective, you may also boost your performance as follows:

- Ensure that the JDBC data source used for the database work exists on the same server instance as the JMS destination—the transaction will still be two-phase, but it will be handled with less network overhead.
- Use file stores rather than JDBC stores.
- Configure multiple services to share the same store if they will commonly be invoked within the same transaction.
- If an application directly performs database operations in addition to invoking store services (such as JMS) within the same transaction, consider using a Logging Last Resource JDBC pool for the database operations.

  The transaction will still be two-phase for the store, and will still be "ACID", but the application direct database operations within a transaction will be run in a single local transaction, which, in turn, may greatly improve overall transaction performance.

BETA

Monitoring a Persistent Store

You can monitor statistics for each existing persistent store and for each open store connection.

Monitoring Stores

Each persistent store is represented at runtime by an instance of the `PersistentStoreRuntimeMBean`, which provides the following options.

Table 7-6 Persistent Store Run-time Options

Option	What it does. . .
CreateCount	Number of create requests issued to this persistent store.
ReadCount	Number of read requests issued to this persistent store.
UpdateCount	Number of update requests issued by this persistent store.
DeleteCount	Number of delete requests issued by this persistent store.
ObjectCount	Number of objects contained in the persistent store.
Connections	Number of active connections in the persistent store.
PhysicalWriteCount	Number of times the persistent store flushes its data to durable storage.

BETA

Monitoring Store Connections

For each open persistent store connection, the persistent store also registers a `PersistentStoreConnectionRuntimemMBean`, which provides the following options.

Table 7-7 Persistent Store Connection Run-time Options

Option	What it does. . .
CreateCount	Number of create requests issued to this connection.
ReadCount	Number of read requests issued to this connection.
UpdateCount	Number of update requests issued by this connection.
DeleteCount	Number of delete requests issued by this connection.
ObjectCount	Number of objects contained in the connection.

Table 7-8 defines the WebLogic services and subsytems that can create connection to the persistent store.

Table 7-8 Persistent Store Users

Subsystem/Service	Runtime Name
JMS Servers	`weblogic.messaging.`*`NAME`* or `weblogic.jms.`*`NAME`* for durable subscribers where *`NAME`* is the name of the JMS server connection to the store
Store-and-Forward Service Agents	`weblogic.messaging.`*`NAME`* where *`NAME`* is the name of the SAF Agent's connection to the store
Path Service	TBD
Web Services	TBD
EJB Timer Services	TBD
Transaction Log (TLOG)	TBD

BETA

Limitations of the Persistent Store

The following limitations apply to the persistent store:

- A persistent file store should not be opened simultaneously by two server instances; otherwise, there is no guarantee that the data in the file will not be corrupted. If possible, the persistent store will attempt to return an error in this case, but it will not be possible to detect this condition in every case. It is the responsibility of the administrator to ensure that the persistent store is being used in an environment in which multiple servers will not try to access the same store at the same time. (Two file stores are considered the "same store" if they have the same name and the same directory.)
- Two JDBC stores must not share the same database table, because this will result in data corruption.
- A persistent store may not survive arbitrary corruption. If the disk file is overwritten with arbitrary data, then the results are undefined. The store may return inconsistent data in this case, or even fail to recover at all.
- A file store may return exceptions when its disk is full. However, it will resume normal operation by no longer throwing an exception when disk space has been made available. Also, the data in the persistent store must remain intact as described in the previous points.

BETA

BETA

Index

BETA

BETA