Copyright

Copyright © 2002 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


All other trademarks are the property of their respective companies.

Programming WebLogic Management Services with JMX

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Document Revised</th>
<th>Software Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>August 23, 2002</td>
<td>BEA WebLogic Server Version 8.1 Beta</td>
</tr>
</tbody>
</table>
Contents

About This Document

Audience............................................................................................................ viii
e-docs Web Site................................................................................................. viii
How to Print the Document............................................................................... viii
Related Information............................................................................................. ix
Contact Us!.......................................................................................................... ix
Documentation Conventions .................................................................................x

1. Overview of WebLogic JMX Services

WebLogic Server Managed Resources and MBeans ................................. 1-2
   Basic Organization of a WebLogic Server Domain.............................. 1-3
   MBeans for Configuring Managed Resources ...................................... 1-4
      Replicating Configuration MBeans ................................................. 1-5
      The Lifecycle of Configuration MBeans ....................................... 1-8
      Replication of MBeans for Managed Server Independence .......... 1-11
      Documentation for Configuration MBean APIs ......................... 1-11
   MBeans for Viewing the Runtime State of Managed Resources ........ 1-12
      Documentation for Runtime MBean APIs .................................. 1-14
   Security MBeans ................................................................................. 1-15
   Non-WebLogic Server MBeans .......................................................... 1-16
   MBean Servers and the MBeanHome Interface .................................. 1-16
      Local MBeanHome and the Administration MBeanHome ........... 1-18
   Notifications and Monitoring ............................................................... 1-20
   The Administration Console and the weblogic.Admin Utility .......... 1-20
      The Administration Console ......................................................... 1-21
      The weblogic.Admin Utility ......................................................... 1-23
2. **Accessing WebLogic Server MBeans**

- Main Steps for Accessing MBeans ................................................................. 2-2
- Determining Which Interfaces to Use ............................................................ 2-2
- Accessing an MBeanHome Interface ............................................................ 2-4
  - Using the Helper APIs to Retrieve an MBeanHome Interface .................... 2-4
  - Example: Retrieving a Local MBeanHome Interface ............................... 2-5
- Using JNDI to Retrieve an MBeanHome Interface ........................................ 2-5
  - Example: Retrieving the Administration MBeanHome from an External Client .............................................................................................................. 2-7
  - Example: Retrieving a Local MBeanHome from an Internal Client .......... 2-9
- Using the MBeanServer Interface to Access MBeans .................................... 2-10
- Using the Type-Safe Interface to Access MBeans ........................................ 2-12
  - Retrieving a List of All MBeans ............................................................... 2-12
  - Retrieving MBeans By Type and Selecting From the List................................ 2-13
- WebLogicObjectNames for WebLogic Server MBeans ............................... 2-16
  - Using weblogic.Admin to Find the WebLogicObjectName .................... 2-19

3. **Accessing and Changing Configuration Information**

- Example: Using weblogic.Admin to Configure the Message Level for Standard Out .............................................................................................................. 3-3
- Example: Using MBeanServer to Configure the Message Level for Standard Out 3-4
- Example: Using the Type-Safe Interface to Retrieve Information About a JMS Configuration .................................................................................................... 3-6

4. **Using WebLogic Server MBean Notifications**

- WebLogic Server Notification Types ............................................................. 4-2
- WebLogic Server Log Notifications ............................................................... 4-2
- Listening for Notifications: Main Steps ......................................................... 4-3
  - Creating a Notification Listener ............................................................... 4-4
  - Registering a Notification Listener .......................................................... 4-6

5. **Accessing Runtime Information**

- Determining the Active Domain and Servers .............................................. 5-2
  - Using weblogic.Admin to Determine Active Domains and Servers .......... 5-4
Example: Viewing and Changing the Runtime State of a WebLogic Server Instance

Using a Local MBeanHome and getRuntimeMBean() ................................. 5-6
Using the Administration MBeanHome and getMBeansByType() .......... 5-9
Using the Administration MBeanHome and getMBean()....................... 5-11
Using the MBeanServer Interface ......................................................... 5-14

Example: Viewing Runtime Information About Clusters ......................... 5-16

6. Monitoring WebLogic Server MBeans

Creating a Notification Listener .............................................................. 6-2
Choosing a WebLogic Server Monitor Type .......................................... 6-3
Instantiating the Monitor and Listener ................................................... 6-5
Main Steps for Instantiating a Monitor and Listener .............................. 6-5
Example: Instantiating a CounterMonitor for a Remote Application ...... 6-7
Configuring CounterMonitor Objects .................................................... 6-9
Configuring GaugeMonitor Objects .................................................... 6-11
Configuring StringMonitor Objects ..................................................... 6-12
Notification Types .............................................................................. 6-13
Error Notification Types .................................................................... 6-14
Sample Monitoring Scenarios .............................................................. 6-14
JDBC Monitoring ............................................................................. 6-15
About This Document

This document describes how to use the BEA WebLogic Server™ management APIs to configure and monitor WebLogic Server domains, clusters, and server instances.

The document is organized as follows:

- **Chapter 1, “Overview of WebLogic JMX Services,”** which describes the WebLogic Server management interface and provides overviews of WebLogic Server MBeans, MBean home interfaces, and the distributed management architecture.

- **Chapter 2, “Accessing WebLogic Server MBeans,”** which describes how to access WebLogic Server MBeans from a client application.

- **Chapter 3, “Accessing and Changing Configuration Information,”** which provides examples for retrieving and modifying the configuration of WebLogic Server resources.

- **Chapter 4, “Using WebLogic Server MBean Notifications,”** which describes how to listen and respond to WebLogic Server MBean notifications in a client application.

- **Chapter 5, “Accessing Runtime Information,”** which provides examples for retrieving and modifying runtime information about WebLogic Server domains and server instances.

- **Chapter 6, “Monitoring WebLogic Server MBeans,”** which describes how to monitor WebLogic Server MBean attributes from a monitor MBean.

**Note:** The WebLogic Security Service provides MBeans and tools for generating additional MBeans that manage security on a WebLogic Server. These MBeans are called Security MBeans and their usage model is different from the one described in this document. For information on Security MBeans, refer to the *Developing Security Services for WebLogic Server* guide.
Audience

This document is written for independent software vendors (ISVs) and other developers who are interested in creating custom applications that use BEA WebLogic Server facilities to monitor and configure applications and server instances. It assumes that you are familiar with the BEA WebLogic Server platform and the Java programming language, but not necessarily with Java Management Extensions (JMX).

While the document describes how to access and use the Managed Beans (MBeans) that WebLogic Server provides, it does not describe how to create your own, additional MBeans. For information about creating and using MBeans in addition to the ones that WebLogic Server provides, refer to the JMX 1.0 specification, which you can download from http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation.

How to Print the Document

You can print a copy of this document from a Web browser, one main topic at a time, by using the File→Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click Download Documentation, and select the document you want to print.

Related Information

The BEA corporate Web site provides all documentation for WebLogic Server. The following BEA WebLogic Server documentation contains information that is relevant to understanding how to use the WebLogic Server management services.

- BEA WebLogic Server Documentation (available online):
  - Administration Guide
  - Programming Guides
  - WebLogic Server API

- The Sun Microsystems, Inc. Java site at http://java.sun.com/

- The JMX 1.0 specification and API documentation at http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html

Contact Us!

Your feedback on BEA documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at http://www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
Your machine type and authorization codes
The name and version of the product you are using
A description of the problem and the content of pertinent error messages

**Documentation Conventions**

The following documentation conventions are used throughout this document.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface text</strong></td>
<td>Indicates terms defined in the glossary.</td>
</tr>
<tr>
<td>Ctrl+Tab</td>
<td>Indicates that you must press two or more keys simultaneously.</td>
</tr>
<tr>
<td><em>italics</em></td>
<td>Indicates emphasis or book titles.</td>
</tr>
</tbody>
</table>
| **monospace text**| Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.  
*Examples:*  
`#include <iostream.h> void main ( ) the pointer psz  
chmod u+w *  
\tux\data\ap  
.doc  
tux.doc  
BITMAP  
float`  
| **monospace boldface text** | Identifies significant words in code.  
*Example:*  
`void commit ( )`  
| **monospace italic text** | Identifies variables in code.  
*Example:*  
`String expr`  

x Programming WebLogic Management Services with JMX
<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
</tr>
</thead>
</table>
| UPPERCASE TEXT | Indicates device names, environment variables, and logical operators.  
**Examples:**  
LPT1  
SIGNON  
OR |
| { } | Indicates a set of choices in a syntax line. The braces themselves should never be typed. |
| [ ] | Indicates optional items in a syntax line. The brackets themselves should never be typed.  
**Example:**  
buildobjclient [-v] [-o name ] [-f file-list]...  
[-l file-list]... |
| | Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed. |
| ... | Indicates one of the following in a command line:  
- That an argument can be repeated several times in a command line  
- That the statement omits additional optional arguments  
- That you can enter additional parameters, values, or other information  
The ellipsis itself should never be typed.  
**Example:**  
buildobjclient [-v] [-o name ] [-f file-list]...  
[-l file-list]... |
| . | Indicates the omission of items from a code example or from a syntax line.  
. The vertical ellipsis itself should never be typed. |
1 Overview of WebLogic JMX Services

To provide open and extensible management services, WebLogic Server implements the Sun Microsystems, Inc. Java Management Extensions (JMX) 1.0 specification. All WebLogic Server resources are managed through these JMX-based services, and third-party services and applications that run within WebLogic Server can be managed through them as well.

The WebLogic Server Administration Console and the weblogic.Admin utility use WebLogic JMX APIs to implement their management services. You can also use these APIs to build your own, specialized management utilities. For example, you can build a management utility that uses JMX APIs to monitor your application’s use of JDBC connection pools. If usage falls outside a set of allowable parameters, your utility can use the APIs to adjust the size or configuration of the connection pools. Your utility could also include code that sends an email to alert a system administrator of the configuration change.

WebLogic Server implements the JMX 1.0 specification and adds its own set of convenience methods and other extensions to take advantage of the WebLogic Server distributed environment. This topic provides an overview of the WebLogic Server JMX services:

- “WebLogic Server Managed Resources and MBeans” on page 1-2
- “MBean Servers and the MBeanHome Interface” on page 1-16
- “Notifications and Monitoring” on page 1-20
- “The Administration Console and the weblogic.Admin Utility” on page 1-20
1 Overview of WebLogic JMX Services

To view the JMX 1.0 specification, download it from http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html. The API documentation is included in the archive that you download.

WebLogic Server Managed Resources and MBeans

Subsystems within WebLogic Server (such as JMS Provider and JDBC Container) and the items that they control (such as JMS servers and JDBC connection pools) are called WebLogic Server managed resources. Each managed resource includes a set of attributes that can be configured and monitored for management purposes. For example, each JDBC connection pool includes attributes that define its name, the name of its driver, its initial capacity, and its cache size. Some managed resources provide additional methods (operations) that can be used for management purposes. The WebLogic JMX services expose these management attributes and operations through one or more managed beans (MBeans). An MBean is a concrete Java class that is developed per JMX specifications. It can provide getter and setter operations for each management attribute within a managed resource along with additional management operations that the resource makes available. (See Figure 1-1.)

Figure 1-1 Managed Resources and Managed Beans

<table>
<thead>
<tr>
<th>WebLogic Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managed Resource</td>
</tr>
<tr>
<td>ManagementAttribute-A</td>
</tr>
<tr>
<td>ManagementAttribute-B</td>
</tr>
<tr>
<td>ManagementOperation-A</td>
</tr>
<tr>
<td>Managed Bean</td>
</tr>
<tr>
<td>getManagementAttribute-A</td>
</tr>
<tr>
<td>setManagementAttribute-A</td>
</tr>
<tr>
<td>getManagementAttribute-B</td>
</tr>
<tr>
<td>setManagementAttribute-B</td>
</tr>
<tr>
<td>ManagementOperation-A</td>
</tr>
</tbody>
</table>
WebLogic Server Managed Resources and MBeans

WebLogic Server MBeans that expose attributes and operations for configuration a managed resource are called Configuration MBeans while MBeans that provide information about the runtime state of a managed resource are called Runtime MBeans. The functions of configuring resources and viewing data about the runtime state of resources are sufficiently different in a WebLogic Server domain that Configuration MBeans and Runtime MBeans are distributed and maintained differently.

This section contains the following subsections:

- “Basic Organization of a WebLogic Server Domain” on page 1-3
- “MBeans for Configuring Managed Resources” on page 4
- “MBeans for Viewing the Runtime State of Managed Resources” on page 1-12
- “Security MBeans” on page 1-15
- “Non-WebLogic Server MBeans” on page 1-16

Basic Organization of a WebLogic Server Domain

A WebLogic Server administration domain is a logically related group of WebLogic Server resources. Domains include a special WebLogic Server instance called the Administration Server, which is the central point from which you configure and manage all resources in the domain. Usually, you configure a domain to include additional WebLogic Server instances called Managed Servers. You deploy applications, EJBs, and other resources that you develop onto the Managed Servers and use the Administration Server for configuration and management purposes only.

Using multiple Managed Servers enables you to balance loads and provide failover protection for critical applications, while using single Administration Server simplifies the management of the Managed Server instances. For more information about domains, refer to "Overview of WebLogic System Administration" in the WebLogic Server Administration Guide.
1 Overview of WebLogic JMX Services

MBeans for Configuring Managed Resources

To support the WebLogic Server model of centralizing management responsibilities on the Administration Server, the Administration Server hosts Configuration MBeans for all managed resources on all server instances in the domain. In addition, the Administration Server saves changes to configuration data so that it is available when you shutdown and restart a server instance.
This section contains the following subsections:

- “Replicating Configuration MBeans” on page 1-5
- “The Lifecycle of Configuration MBeans” on page 1-8
- “Replication of MBeans for Managed Server Independence” on page 1-11
- “Documentation for Configuration MBean APIs” on page 1-11

**Replicating Configuration MBeans**

To enhance performance and enable flexibility when configuring distributed resources, each Managed Server creates local replicas of the Configuration MBeans for its own managed resources. Objects that interact with MBeans (MBean clients) use the replicas on the local server instead of initiating remote calls to the Administration Server. (See Figure 1-2.)
The Configuration MBeans on the Administration Server are called **Administration MBeans**, and the replicas on the Managed Servers are called **Local Configuration MBeans**.

In addition to enhancing performance, this distribution of MBeans enables you to choose between making permanent or temporary changes to a server’s configuration:

- Changes that you make to an Administration MBean are saved in a configuration file called `config.xml` and are available across server sessions. You can change an Administration MBean through the Administration Console, the `weblogic.Admin` command utility, or by using one of the MBean’s setter methods.

- Changes that you make to a Local Configuration MBean are not saved to `config.xml` and apply only to the current server session. You can change a Local Configuration MBean through the `weblogic.Admin` command utility, by using one of the MBean’s setter methods, or by using options of the
The `weblogic.Server` startup command. The `weblogic.Server` options modify the values in the Local Configuration MBean, overriding the values from the Administration MBean. You cannot use the Administration Console to view or modify Local Configuration MBeans.

While the Administration Server always hosts Administration MBeans, it can potentially host Local Configuration MBeans as well. For example, if you set up managed resources on the Administration Server, it must host both the Administration MBeans for all server instances in the domain along with the Local Configuration MBeans associated with the resource. MBean clients on the Administration Server use the Local Configuration MBean. (See Figure 1-3.)

**Figure 1-3 Local Configuration MBean on Administration Server**

![Diagram showing local configuration MBean on administration server](image-url)
The Lifecycle of Configuration MBeans

This section describes how Administration MBeans and Local Configuration MBeans are initialized, how changes to configuration data is propagated throughout the WebLogic Server system, and how attribute values can be changed so that they are available when you restart server instances:

1. The lifecycle of a Configuration MBean begins when you start the Administration Server. During its startup cycle, the Administration Server initializes all the Administration MBeans for the domain with data from the domain’s config.xml file. (See Figure 1-4.)

   Figure 1-4  Initializing Configuration MBeans

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <Domain>
     <Server
       Name="examplesServer"
       ListenPort="7001"
     >
     </Server>
   </Domain>
   ```

   The Administration Server reads data from the config.xml file only during its startup cycle.

2. When a Managed Server starts, it contacts the Administration Server for its configuration data. By default, it creates replicas of the Administration MBeans that configure its local resources. However, you can use arguments in the server’s startup command to override values of the Administration MBeans.

   For example, for Managed Server A, the config.xml file states that its listen port is 8000. When you use the weblogic.Server command to start Managed Server A, you include the -Dweblogic.ListenPort=7501 startup option to change the listen port for the current server session. The Managed Server creates a replica of the Administration MBeans, but substitutes 7501 as the value of its...
listen port. When you restart Managed Server A, it will revert to using the value from the config.xml file, 8000. (See Figure 1-5.)

Figure 1-5  Overriding Administration MBean Values

1. At startup, the Administration Server initializes Administration MBeans with data from the config.xml file.

2. At startup, Managed Servers replicate the Administration MBeans. Startup options override the values from the Administration MBeans.

weblogic.ListenPort=8000

Managed Server A

Managed Server B

weblogic.ListenPort=7501

weblogic.ListenPort=7501
When you start an Administration Server, any startup command arguments that you use to override the values in `config.xml` affect only the values of the Local Configuration MBeans on the Administration Server. The command arguments do not affect the values of the Administration MBeans and therefore do not affect subsequent server sessions. (See Figure 1-6.)

Figure 1-6 Overriding Values on the Administration Server

3. If you change a value in an Administration MBean, and if the corresponding Managed Server is running, the Administration Server propagates the change to the Local Configuration MBean. Depending on the attribute, the underlying resource might not be able to accept the new value until it restarts. The WebLogic Server Javadoc indicates whether a managed resource can accept new values for an attribute during the current session. Even if a managed resource can accept new values, depending on the frequency with which the resource checks for configuration changes, the resource might not use the updated value immediately.
4. Periodically, the Administration Server determines whether Administration MBeans have been changed and writes any changes back to config.xml. Changes also are written to config.xml when the Administration Server shuts down or when MBean attributes are modified by a WebLogic Server utility such as the Administration Console or weblogic.Admin.

5. Local Configuration MBeans are destroyed when you shut down Managed Servers. Administration MBeans are destroyed when you shut down the Administration Server.

**Replication of MBeans for Managed Server Independence**

Managed Server Independence (MSI) is a feature that enables a Managed Server to start if the Administration Server is unavailable. If a Managed Server is configured for MSI, in addition to its Local Configuration MBeans, it also contains a copy of all Administration MBeans for the domain.

Do not interact with these Administration MBeans on a Managed Server. They reflect the last known configuration for the domain and are used only for starting the Managed Server in MSI mode. Modifying an Administration MBean on a Managed Server can cause the Managed Server’s configuration to be inconsistent with the Administration Server, which will lead to unpredictable results. In addition, Managed Servers are not aware of the Administration MBeans on other Managed Servers.

For more information on MSI, refer to "Starting a Managed Server When the Administration Server Is Not Accessible" in the Configuring and Managing WebLogic Server guide.

**Documentation for Configuration MBean APIs**

To view the documentation for Configuration MBeans, do the following:

1. Open the WebLogic Server Javadoc.

2. In the top left pane of the Web browser, click weblogic.management.configuration.

   The lower left pane displays links for the package.
Overview of WebLogic JMX Services

3. In the lower left pane, click `weblogic.management.configuration` again.

The right pane displays the package summary. (See Figure 1-7.)

Figure 1-7 Javadoc for the configuration Package

4. Click on an interface name to view its API documentation.

MBeans for Viewing the Runtime State of Managed Resources

WebLogic Server managed resources provide performance metrics and other information about their runtime state through one or more Runtime MBeans. Runtime MBeans are not replicated like Configuration MBeans, and they exist only on the same server instance as their underlying managed resources.

Because Runtime MBeans contain only transient data, they do not save their data in the `config.xml` file. When you shut down a server instance, all runtime statistics and metrics from the Runtime MBeans are destroyed.
The following figure (Figure 1-8) illustrates how Runtime MBeans, Administration MBeans, and Local Configuration MBeans are distributed throughout a domain.

**Figure 1-8 Distribution of MBeans**
1 Overview of WebLogic JMX Services

You can use the Administration Console, the `weblogic.Admin` utility, or MBean APIs to view the values. (See Figure 1-9.)

Figure 1-9 Viewing Runtime Metrics from the Administration Console

You can also use these interfaces to change some runtime values. For example, the `weblogic.management.runtime.DeployerRuntimeMBean` activates and deactivates a deployed module by changing its runtime state.

Documentation for Runtime MBean APIs

To view the documentation for Runtime MBeans, do the following:

1. Open the WebLogic Server Javadoc.
2. In the top left pane of the Web browser, click `weblogic.management.runtime`. The lower left pane displays links for the package.
3. In the lower left pane, click `weblogic.management.runtime` again.

The right pane displays the package summary. (See Figure 1-10.)

**Figure 1-10  Javadoc for the runtime Package**

3. In the lower left pane, click `weblogic.management.runtime` again.

The right pane displays the package summary. (See Figure 1-10.)

4. Click on an interface name to view its API documentation.

**Security MBeans**

The WebLogic Security Service provides MBeans and tools for generating additional MBeans that manage security on a WebLogic Server. These MBeans are called Security MBeans and their usage model is different from the one described in this document. For information on Security MBeans, refer to the *Developing Security Services for WebLogic Server* guide.
Non-WebLogic Server MBeans

WebLogic Server provides hundreds of MBeans, many of which are used to configure and monitor EJBs, Web applications, and other deployable J2EE modules. If you want to use additional MBeans to configure your applications or services, you can create your own MBeans.

Any MBeans that you create can take advantage of the full set of JMX 1.0 features, as defined by the JMX specification (which you can download from [http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html](http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html)).

However, only MBeans that are provided by WebLogic Server can use the WebLogic Server extensions to JMX. For example, any MBeans that you create for your applications cannot save data in the config.xml file and they cannot use the type-safe interface as described in the next section, “MBean Servers and the MBeanHome Interface.”

MBean Servers and the MBeanHome Interface

Within a WebLogic Server instance, the actual work of registering and providing access to MBeans is delegated to an MBean Server subsystem. The MBean Server on a Managed Server registers and provides access only to the Local Configuration MBeans and Runtime MBeans on the current Managed Server. The MBean Server on an Administration Server registers and provides access to the domain’s Administration MBeans as well as the Local Configuration MBeans and Runtime MBeans on the Administration Server.

**Note:** On a Managed Server that is configured for MSI, the MBean Server also registers the Administration MBean replicas that the server uses to start if the Administration Server is not available. Do not interact with these Administration MBean replicas. For more information, refer to “Replication of MBeans for Managed Server Independence” on page 1-11.
To access the MBean Server subsystem, you use the
weblogic.management.MBeanHome interface. From MBeanHome, you can use any of
the following interfaces to interact with the MBean Server and its MBeans (see
Figure 1-11):

- javax.management.MBeanServer, which is the standard JMX interface for
  interacting with MBeans. You can use this interface to look up MBeans that are
  registered in an MBean Server, determine the set of operations available for an
  MBean, and determine the type of data that each operation returns. If you invoke
  MBean operations through the MBeanServer interface, you must use standard
  JMX methods. For example:

  ```java
  MBeanHome.getMBeanServer.getAttribute(MBeanObjectName, attributeName)
  MBeanHome.getMBeanServer.setAttribute(MBeanObjectName, attributeName)
  MBeanHome.getMBeanServer.invoke(MBeanObjectName, operationName, params, signature)
  ```

  For a complete list of MBeanServer APIs, refer to view the JMX 1.0 API
documentation, which you can download from
http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html. The archive
that you download includes the API documentation.

  The MBeanServer interface is your only option for interacting with MBeans that
  you have created and registered (non-WebLogic MBeans).

- A WebLogic Server type-safe interface that makes it appear as though you can
  invoke an MBean’s methods directly. You can use this interface to look up
  MBeans that are registered in an MBean Server and invoke get, set, and other
  operations on the MBean. For example:

  ```java
  wlMBean = MBeanHome.getMBean(WebLogicObjectName)
  wlMBean.getAttribute
  wlMBean.setAttribute
  wlMBean.invoke
  ```
Local MBeanHome and the Administration MBeanHome

All instances of WebLogic Server provide a **local MBeanHome** interface through which you can access the MBeans that are hosted in the server instance’s MBean Server.

For Managed Servers and Administration Servers, the local MBeanHome interface provides access to the Local Configuration MBeans and Runtime MBeans for the current server only.

The Administration Server provides an additional instance of the MBeanHome interface. This **Administration MBeanHome** provides access to Administration MBeans along with all other MBeans on all server instances in the domain. While the Administration MBeanHome provides a single access point for all MBeans in the domain, you must sort through the lookup results to find an MBean for a specific WebLogic Server instance. In addition, it uses RMI to contact MBeans on Managed Servers, which uses more network resources and might take longer than using a local MBeanServer or MBeanHome interface. (See Figure 1-12.)
The local `MBeanHome` and the Administration `MBeanHome` are two instances of the same interface class, so the APIs for the two types of `MBeanHome` differ only in the name of the `MBeanHome` instance and in the set of `MBeans` that you can access.
Notifications and Monitoring

Depending on your management needs, you can use MBean APIs to view MBean attributes only upon request, or you can use the WebLogic Server notification and monitoring facilities, which automatically broadcast reports (JMX notifications) when MBean attributes change.

To use these facilities, you can do the following:

- Create a JMX listener, which listens for and reports all attribute changes within an MBean that you specify. For example, you could use a listener with some additional logic to send an email to a System Administrator any time a user changes the configuration of a deployed component. For information about using listeners, refer to Chapter 4, “Using WebLogic Server MBean Notifications.”

- Create a JMX monitor, which listens for and reports only the changes to specific MBean attributes that fall outside a set of parameters that you set. For example, you could use a monitor with some additional logic to send an email to a System Administrator when the number of open thread pools exceeds a specified limit. For more information, refer to Chapter 6, “Monitoring WebLogic Server MBeans.”

The Administration Console and the weblogic.Admin Utility

Even if you are developing applications to use the WebLogic Server JMX implementation, you will probably use the WebLogic Server Administration Console and the weblogic.Admin utility for some management tasks. In some cases, you might use these interfaces to familiarize yourself with some area of WebLogic Server management services before developing your JMX applications.
This section contains the following subsections:

- “The Administration Console” on page 1-21
- “The weblogic.Admin Utility” on page 1-23

The Administration Console

The Administration Console is a Web application with servlets that invoke the WebLogic Server JMX APIs. Almost all of the values that the Administration Console presents are attributes of Administration MBeans and Runtime MBeans. Because the Administration Console does not read or write Local Configuration MBeans, it is possible that it reports a value that a server instance is not currently using. For example, if you use a weblogic.Server startup option to override the configured listen port, the Administration Console reports the value that is in the config.xml file, not the overriding value.
Overview of WebLogic JMX Services

To determine which MBean attribute the Administration Console is presenting, click the question mark next to a field. A help window displays the associated MBean class, attribute, and Javadoc description. (See Figure 1-13.)

Figure 1-13 Viewing MBean Associations from the Administration Console

The caution icon (yellow triangle with an exclamation point) indicates that an attribute is not dynamic. If you modify such an attribute, the underlying managed resource cannot use the new value until you restart the server.

If you modify a dynamic value from the Administration Console (such as Startup Mode in Figure 1-13), the console updates the corresponding Administration MBean. For information on how this change is propagated to the Local Configuration MBean, refer to “The Lifecycle of Configuration MBeans” on page 1-8.
The Administration Console and the weblogic.Admin Utility

The weblogic.Admin Utility

The weblogic.Admin utility provides several commands that create, get and set values for, invoke operations on, and delete instances of Administration and Configuration MBeans. It also provides commands to get values and invoke operations on Runtime MBeans. You could create shell scripts that use this utility instead of creating JMX applications to programmatically interact with the WebLogic Server management services, however, the performance of a JMX application is superior to a shell script that invoke command-line utilities.

You can also use the weblogic.Admin utility to verify object names of MBeans and to get and set attributes from a command line before committing to writing JMX code. Subsequent sections in this document provide examples of using the weblogic.Admin utility as part of your JMX development.

For more information, refer to "MBean Management Command Reference" in the WebLogic Server Command Line Reference.
Overview of WebLogic JMX Services
CHAPTER

2 Accessing WebLogic Server MBeans

All JMX tasks—viewing or changing MBean attributes, using notifications, and monitoring changes—use the same process for accessing MBeans.

This topic contains the following sections:

- “Main Steps for Accessing MBeans” on page 2-2
- “Determining Which Interfaces to Use” on page 2-2
- “Accessing an MBeanHome Interface” on page 2-4
- “Using the MBeanServer Interface to Access MBeans” on page 2-10
- “Using the Type-Safe Interface to Access MBeans” on page 2-12
- “WebLogicObjectNames for WebLogic Server MBeans” on page 2-16
Main Steps for Accessing MBeans

The main steps for accessing MBeans in WebLogic Server are as follows:

1. Use a `weblogic.management.MBeanHome` interface to access the MBean Server.
   
   You can use the local MBeanHome interface from any instance of WebLogic Server to access the MBeans that are registered and active on the current server instance. If you want access to all MBeans in the domain, you can use the Administration MBeanHome interface on the Administration Server instead of the local MBeanHome interface.

2. Use one of the following interfaces to retrieve, look up, and invoke operations on MBeans:
   
   - The standard JMX `javax.management.MBeanServer` interface, which can retrieve and invoke operations on WebLogic Server MBeans or on MBeans that you create.
   
   - A type-safe interface that WebLogic Server provides. This interface, which is a WebLogic Server extension to JMX, can retrieve and invoke operations only on the MBeans that WebLogic Server provides.

   In most cases, you use these interfaces to retrieve a list of MBeans and then filter the list to retrieve and invoke operations on a specific MBean. However, if you know the WebLogicObjectName of an MBean, you can retrieve an MBean directly by name.

Determining Which Interfaces to Use

When accessing MBeans, you must make two choices about which interfaces you use:

1. Whether to use the MBeanHome interface on a local server instance or the Administration MBeanHome to access the MBean Server. The MBeanHome interface that you choose determines the set of MBeans you can access.

   The following table lists typical considerations for determining whether to use the local MBeanHome interface or the Administration MBeanHome interface.
Determining Which Interfaces to Use

### If your application manages... | Retrieve this `MBeanHome` interface...
---|---
Administration MBeans | Administration `MBeanHome`
Multiple WebLogic Server instances in a domain | Administration `MBeanHome`
A single WebLogic Server instance in a domain | Local `MBeanHome`

Using the local interface saves you the trouble of filtering MBeans to find those that apply to the single server. Using the local interface also uses fewer network hops to access MBeans, because you are connecting directly to the Managed Server.

2. Whether to use the standard JMX `MBeanServer` interface or the WebLogic Server type-safe interface to access and invoke operations on MBeans.

The following table lists typical considerations for determining whether to use the `MBeanServer` interface or the type-safe interface.

### If your application... | Use this interface...
---|---
Interacts only with WebLogic Server MBeans. | The WebLogic Server type-safe interface
Might need to run on J2EE platforms other than WebLogic Server | `MBeanServer`
Interacts with non-WebLogic Server MBeans | `MBeanServer`
Accessing an MBeanHome Interface

The simplest process for retrieving a local MBeanHome interface or an Administration MBeanHome interface is to use the WebLogic Server Helper class. If you are more comfortable with a standard J2EE approach, you can use the Java Naming and Directory Interface (JNDI) to retrieve MBeanHome.

This section contains the following subsections:

- Using the Helper APIs to Retrieve an MBeanHome Interface
- Using JNDI to Retrieve an MBeanHome Interface

Using the Helper APIs to Retrieve an MBeanHome Interface

WebLogic Server provides the `weblogic.management.Helper` APIs to simplify the process of retrieving MBeanHome interfaces.

To use the Helper APIs, collect the following information:

- The username and password of a user who has permission to invoke MBean operations.
- If you are accessing a local MBeanHome interface, the name of the target server (as defined in the domain configuration) and the URL of the target server.
- If you are accessing the Administration MBeanHome, the URL of the Administration Server.

After you collect the information, use one of the following APIs:

- To retrieve a local MBeanHome:
  ```java
  Helper.getMBeanHome(java.lang.String user, java.lang.String password, java.lang.String serverURL, java.lang.String serverName)
  ```
To retrieve the Administration MBeanHome:

```java
Helper.getAdminMBeanHome(java.lang.String user,
   java.lang.String password, java.lang.String adminServerURL)
```

For more information about the APIs, refer to the `Helper Javadoc`

### Example: Retrieving a Local MBeanHome Interface

The following example (Listing 2-1) is a class that uses the `Helper API` to obtain the local MBeanHome interface for a server named `peach`.

#### Listing 2-1 Retrieving a Local MBeanHome Interface

```java
public void find(String host,
   int port,
   String username
   String password){
   String url = "t3://" + host + ":" + port;
   try {
      localHome = (MBeanHome)Helper.getMBeanHome(username,
                                               password,
                                               url,
                                               "peach");
      System.out.println("Local MBean Home " +
                        "found using the Helper class");
   } catch (IllegalArgumentException iae) {
      System.out.println("Illegal Argument Exception: " + iae);
   }
}
```

### Using JNDI to Retrieve an MBeanHome Interface

While the `Helper APIs` provide a simple way to obtain an MBeanHome interface, you might be more familiar with the standard approach of using JNDI to retrieve the MBeanHome. From the JNDI tree of a Managed Server, you can access the server's
Accessing WebLogic Server MBeans

local MBeanHome interface. From the JNDI tree of the Administration Server, you can access the Administration MBeanHome as well as the local MBeanHome interface for any server instance in the domain.

To use JNDI to retrieve an MBeanHome interface, do the following:

1. Use weblogic.jndi.Environment methods to set an initial context.

   If your application and the MBeanHome that you want to retrieve are running in the same JVM, the following API is sufficient for setting an initial context:
   
   Environment.getInitialContext()

   If your application and the MBeanHome are in different JVMs, you must use Environment methods to set the initial context with the following properties:

   - The URL of the WebLogic Server instance that hosts the MBeanHome you want to retrieve.
   - The username and password of a user who has permission to access the MBean.

   For example, the following lines of code set the initial context to a host named peach:

   ```java
   Environment env = new Environment();
   env.setProviderUrl("t3://peach:7001");
   env.setSecurityPrincipal("weblogic");
   env.setSecurityCredentials("weblogic");
   Context ctx = env.getInitialContext();
   ```

   For more information about weblogic.jndi.Environment, refer to the WebLogic Server Javadoc.

2. Use javax.naming.Context.lookup(String name) to retrieve the MBeanHome interface.

   Supply one of the following values for the name argument depending on which MBeanHome interface you are retrieving:

   - To retrieve the local MBeanHome for the current context, use the following value:
     MBeanHome.LOCAL_JNDI_NAME
   - If the current context is an Administration Server, you can supply the following value to retrieve the local MBeanHome of any server instance in the domain:
     weblogic.management.home.relevantServerName
where `relevantServerName` is the name of a server as defined in the domain configuration.

- If the current context is an Administration Server, you can supply the following value to retrieve the Administration MBean Home:
  ```
  MBeanHome.ADMIN_JNDI_NAME
  ```

The Administration MBeanHome interface provides access to all Local Configuration, Administration, and Runtime MBeans in the domain.

For more information about `javax.naming.Context.lookup(String name)`, refer to the JNDI Javadoc.

The following sections provide examples for retrieving MBeanHome interfaces:

- **Example: Retrieving the Administration MBeanHome from an External Client**
- **Example: Retrieving a Local MBeanHome from an Internal Client**

### Example: Retrieving the Administration MBeanHome from an External Client

The following example (Listing 2-2) shows how an application running in a separate JVM would look up the Administration MBeanHome interface. In the example, `weblogic` is a user who has permission to view and modify MBean attributes. For information about permissions to view and modify MBeans, refer to "Protecting System Administration Operations" in the WebLogic Server Administration Guide.

#### Listing 2-2  Retrieving the Administration MBeanHome from an External Client

```java
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.AuthenticationException;
import javax.naming.CommunicationException;
import javax.naming.NamingException;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
...

public static void main(String[] args) {
    MBeanHome home = null;
```
// domain variables
String url = "t3://localhost:7001";
String username = "weblogic";
String password = "weblogic";

// Setting an initial context.
try {
    Environment env = new Environment();
    env.setProviderUrl(url);
    env.setSecurityPrincipal(username);
    env.setSecurityCredentials(password);
    Context ctx = env.getInitialContext();

    // Retrieving the Administration MBeanHome interface
    home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
    System.out.println("Got the Admin MBeanHome: " + home + " from the Admin server");
} catch (Exception e) {
    System.out.println("Exception caught: " + e);
}
Example: Retrieving a Local MBeanHome from an Internal Client

If your client application resides in the same JVM as the Administration Server (or the WebLogic Server instance you want to manage), the JNDI lookup for the MBeanHome is simpler. Listing 2-3 shows how a servlet running in the same JVM as the Administration Server would look up the local MBeanHome for a server instance named melon.

Listing 2-3 Retrieving a Local MBeanHome from an Internal Client

```java
import java.io.PrintWriter;
import java.io.IOException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import weblogic.logging.NonCatalogLogger;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import javax.naming.Context;
public class MyServlet extends HttpServlet {
    public void findInternal() {
        Environment env = new Environment();
        try {
            //Setting the initial context
            ctx = env.getInitialContext();
            //Retrieving the server-specific MBeanHome interface
            home = (MBeanHome)ctx.lookup(weblogic.management.home.melon);
            System.out.println("Got the Server-specific MBeanHome: " + home);
        } catch (Exception e) {
            System.out.println("Exception caught: " + e);
        }
    }
}
```
Using the MBeanServer Interface to Access MBeans

A standard JMX approach for interacting with MBeans is to use the javax.management.MBeanServer interface to look up MBeans that are within the scope of the MBeanHome interface. Then you use the MBeanServer interface to get or set MBean attributes or to invoke MBean operations.

The example class in Listing 2-4 uses JNDI to retrieve the Administration MBeanHome interface. Then it retrieves the MBeanServer interface and uses a query to look up all instances of JDBCConnectionPoolMBean in the domain.

For the complete list of MBeanServer methods, refer to the JMX 1.0 API documentation, which you can download from http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html. The archive that you download includes the API documentation.

Listing 2-4 Using the MBeanServer Interface

```java
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.AuthenticationException;
import javax.naming.CommunicationException;
import javax.naming.NamingException;
import javax.management.MBeanServer;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.RemoteMBeanServer;
...

public static void main(String[] args) {
  MBeanHome home = null;
  RemoteMBeanServer homeServer = null;

  //domain variables
  String url = "t3://localhost:7001";

  String username = "weblogic";
  String password = "weblogic";
```
// Setting an initial context.
try {
    Environment env = new Environment();
    env.setProviderUrl(url);
    env.setSecurityPrincipal(username);
    env.setSecurityCredentials(password);
    Context ctx = env.getInitialContext();

    // Retrieving the Administration MBeanHome interface
    home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
    System.out.println("Got the Admin MBeanHome: " + home + " from the Admin
server");

    } catch (Exception e) {
    System.out.println("Exception caught: " + e);

    // Retrieving the MBeanServer interface
    homeServer = home.getMBeanServer();

    // Retrieving a list of MBeans with object names that include
    // "JDBCConnectionPool"
    java.util.Set JDBCMBeans = homeServer.queryNames(new
        ObjectName("mydomain:Type=JDBCConnectionPool,*"), query);
    // where "query" could be any object that implements the JMX
    // java.managementQueryExp
    for (Iterator itr = JDBCMBeans.iterator(); itr.hasNext(); ) {
        WebLogicMBean mbean = (WebLogicMBean)itr.next();
        System.out.println("Matches to the MBean query: " + mbean);
    }
}
Using the Type-Safe Interface to Access MBeans

A simpler approach for accessing MBeans is to use methods of the \( \text{MBeanHome} \) interface. These methods look up WebLogic Server MBeans and return a type-safe interface that you can use to get and set attributes and invoke MBean operations.

This section contains the following subsections:
- “Retrieving a List of All MBeans” on page 2-12
- “Retrieving MBeans By Type and Selecting From the List” on page 2-13

Retrieving a List of All MBeans

You can use the \( \text{MBeanHome.getAllMBeans} \) method to look up the object names of MBeans that are within the scope of the \( \text{MBeanHome} \) interface that you retrieve. For example, if you retrieve the Administrative \( \text{MBeanHome} \), using \( \text{getAllMBeans()} \) returns a list of all MBeans in the domain.
The example code in Listing 2-5 retrieves all MBeans in the domain. It then uses `weblogic.management.WebLogicMBean.getName()` to retrieve the Name value of the `WebLogicObjectName`.

Listing 2-5  Retrieving All MBeans in a Domain

```java
public void displayMBeans() {
    Set allMBeans = home.getAllMBeans();
    System.out.println("Size: "+ allMBeans.size());
    for (Iterator itr = allMBeans.iterator(); itr.hasNext(); ) {
        WebLogicMBean mbean = (WebLogicMBean)itr.next();
        WebLogicObjectName objectName = mbean.getObjectName();
        System.out.println(objectName.getName() + " is a(n) " + mbean.getType());
    }
}
```

For more information about the `MBeanHome.getAllMBeans` method, refer to the `WebLogic Server Javadoc`.

Retrieving MBeans By Type and Selecting From the List

Instead of retrieving a list of all MBeans in the scope of `MBeanHome`, you can retrieve only the list of MBeans that match a specific type. **Type** indicates the type of resource that the MBean manages and whether the MBean is an Administration, Local Configuration, or Runtime MBean. For more information about types of MBeans, refer to the next section, “WebLogicObjectNames for WebLogic Server MBeans” on page 2-16.
The example class in **Listing 2-6** retrieves a list of all `ServerRuntime` MBeans in a domain, and then iterates through the list to select the `ServerRuntime` for a server named `Server1`.

**Listing 2-6 Selecting from a List of MBeans**

```java
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.*;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import javax.management.ObjectName;
import weblogic.management.WebLogicMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.runtime.ServerRuntimeMBean;
import weblogic.management.WebLogicObjectName;

public class serverRuntimeInfo3 {
    public static void main(String[] args) {
        MBeanHome home = null;

        //domain variables
        String url = "t3://localhost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";

        ServerRuntimeMBean serverRuntime = null;
        Set mbeanSet = null;
        Iterator mbeanIterator = null;

        //Setting the initial context
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            //Getting the Administration MBeanHome.
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got the Admin MBeanHome: " + home);
        } catch (Exception e) {
        }
    }
}
```
System.out.println("Exception caught: " + e);
}

/* Here we use the getMBeansByType method to get the set of ServerRuntime mbeans
 * Then we iterate through the set. We retrieve the ServerRuntimeMbean we are
 * interested in by comparing the name to the name of the server.
 */
try {
    mbeanSet = home.getMBeansByType("ServerRuntime");
    mbeanIterator = mbeanSet.iterator();
    while(mbeanIterator.hasNext()) {
        serverRuntime = (ServerRuntimeMBean)mbeanIterator.next();
        if(serverRuntime.getName().equals(serverName)) {
            System.out.println("we have got the serverRuntime mbean: " + serverRuntime +
                           " for: " + serverName);
        }
    }
} for more information about the MBeanHome.getMBeansByType method, refer to
WebLogic Server Javadoc.
WebLogicObjectNames for WebLogic Server MBeans

Each WebLogic Server MBean is registered in the MBean Server under a name that conforms to the `weblogic.management.WebLogicObjectName` conventions. If you know the `WebLogicObjectName` of an MBean, after you retrieve an `MBeanHome` interface, you can retrieve an MBean directly by name.

The MBean’s `WebLogicObjectName` uses the following conventions to provide a unique identification for a given MBean across all domains:

```
domain:Name=name, Type=type[, attr=value]...
```

The following table describes each name component.

<table>
<thead>
<tr>
<th>This Component</th>
<th>Specifies</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>domain</code></td>
<td>The name of the WebLogic Server administration domain.</td>
</tr>
<tr>
<td><code>Name=name</code></td>
<td>The string that you provided when you created the associated resource. For example, when you create a JDBC connection pool, you must provide a name for that pool, such as <code>MyPool1</code>. The <code>JDBCConnectionPoolMBean</code> that represents <code>MyPool1</code> uses <code>Name=MyPool1</code> in its JMX object name. The <code>WebLogicObjectName.getName</code> method returns this value for any given MBean.</td>
</tr>
</tbody>
</table>
WebLogicObjectNames for WebLogic Server MBeans

<table>
<thead>
<tr>
<th>This Component</th>
<th>Specifies</th>
</tr>
</thead>
</table>
| Type=type      | Refers to the interface class of which the MBean is an instance. All WebLogic Server MBeans are an instance of one of the interface classes defined in the weblogic.management.configuration or weblogic.management.runtime packages. For Configuration MBeans, type also refers to whether an instance is an Administration MBean or a Local Configuration MBean. For a complete list of all WebLogic Server MBean interface classes, refer to the WebLogic Server Javadoc for the weblogic.management.configuration or weblogic.management.runtime packages. To determine the value that you provide for the Type component, do the following:
1. Find the MBean’s interface class and remove the MBean suffix from the class name. For example, for an MBean that is an instance of the weblogic.management.runtime.JDBCConnectionPoolRuntimeMBean, use JDBCConnectionPoolRuntime.
2. For a Local Configuration MBean, append Config to the name. For example, for a Local Configuration MBean that is an instance of the weblogic.management.configuration.JDBCConnectionPoolMBean interface class, use JDBCConnectionPoolConfig. For the corresponding Administration MBean instance, use JDBCConnectionPool. |

| Location=servername | All Runtime and Local Configuration MBeans include a Location component that specifies the name of the server on which that MBean is located. Administration MBeans do not include this component. For example, for the ServletRuntime MBean that runs on a server named myserver, the WebLogicObjectName includes the following components: mydomain:Name=myServlet, Type=ServletRuntime, Location=myserver. The WebLogicObjectName.getLocation method returns this value for any given MBean. |
Runtime, Local Configuration, or Administration MBeans that have a child relationship with a parent MBean use this extra attribute in their object names to identify the relationship.

**Note:** With the exception of DomainMBean, all MBeans are direct or indirect children of the domain’s DomainMBean. Because this parent-child relationship applies to all MBeans, it is not expressed in WebLogicObjectName.

For example, an instance of LogMBean is used by a domain to configure the domain-wide log file. Each WebLogic Server instance also maintains its own instance of LogMBean to configure its server-specific log file. The LogMBean that a domain uses does not express a child relationship, while the LogMBean that a server instance uses expresses its child relationship with the server’s ServerMBean. (See Figure 2-1.)

To express the name of the Administration LogMBean that examplesServer uses to maintain its log file, use the following name:

examples:Name=examplesServer,Server=examplesServer,Type=Log

To express the name of the Local Configuration LogMBean that examplesServer uses to maintain its log file, use the following name:

examples:Location=examplesServer,Name=examplesServer,ServerConfig=examplesServer,Type=LogConfig

By convention, WebLogic Server child MBeans use the same value for the Name component as the parent MBean. For example, the LogMBean that is a child of the examplesServer ServerMBean uses Name=examplesServer in its WebLogicObjectName. WebLogic Server cannot follow this convention when a parent MBean has multiple children of the same type.

To determine whether the WebLogicObjectName of an MBean expresses a parent-child relation, use the WebLogicObjectName.getParent method or the weblogic.Admin GET command.
Using `weblogic.Admin` to Find the `WebLogicObjectName`

If you are unsure which values to supply for an MBean’s `WebLogicObjectName`, you can use the `weblogic.Admin` utility to find the `WebLogicObjectName`. The utility can return information only for WebLogic Server MBeans that are on an active server instance.
For example, to find the WebLogicObjectName for the Administration instance of the LogMBean in the examples domain, enter the following command on the examplesServer Administration Server, where the Administration Server is listening on port 8001 and weblogic is the name and password of a user who has permission to view MBean attributes:

```java
```

The command returns the output in Listing 2-7. Notice that the command returns two MBeans of type Log on the Administration Server. The first MBean, examples:Name=examplesServer,Server=examplesServer,Type=Log, has a child relationship with the ServerMBean of examplesServer; this relationship indicates that the MBean is the LogMBean that configures the server-specific log file. The second MBean, examples:Name=examples,Type=Log, has no child relationship, which indicates that it configures the domain-wide log file.

### Listing 2-7  Output from weblogic.Admin

```
MBeanName:  "examples:Name=examplesServer,Server=examplesServer,Type=Log"
  CachingDisabled: true
  FileCount: 7
  FileMinSize: 500
  FileName: examplesServer\examplesServer.log
  FileTimeSpan: 24
  Name: examplesServer
  Notes:
  NumberOfFilesLimited: false
  ObjectName: examplesServer
  Registered: false
  RotationTime: 00:00
  RotationType: none
  Type: Log

MBeanName:  "examples:Name=examples,Type=Log"
  CachingDisabled: true
  FileCount: 7
  FileMinSize: 500
  FileName: ./logs/wl-domain.log
  FileTimeSpan: 24
  Name: examples
  Notes:
```
WebLogicObjectNames for WebLogic Server MBeans

NumberOfFilesLimited: false
ObjectName: examples
Registered: false
RotationTime: 00:00
RotationType: none
Type: Log
Accessing WebLogic Server MBeans

To view the Local Configuration MBean instances of LogMBean, append `Config` to the value of the `type` argument:

```
java weblogic.Admin -url http://localhost:8001 -username weblogic
-password weblogic GET -pretty -type LogConfig
```

The command returns output in Listing 2-8. Notice that the `WebLogicObjectName` of the Local Configuration MBeans includes a `Location` component.

### Listing 2-8 Local Configuration MBeans

```
-------------------------------
MBeanName:
"examples:Location=examplesServer,Name=examplesServer,ServerConfig=examplesServer,Type=LogConfig"
  CachingDisabled: true
  FileCount: 7
  FileMinSize: 500
  FileName: examplesServer\examplesServer.log
  FileTimeSpan: 24
  Name: examplesServer
  Notes:
  NumberOfFilesLimited: false
  ObjectName: examplesServer
  Registered: false
  RotationTime: 00:00
  RotationType: none
  Type: LogConfig
-------------------------------
MBeanName:
"examples:Location=examplesServer,Name=examples,Type=LogConfig"
  CachingDisabled: true
  FileCount: 7
  FileMinSize: 500
  FileName: ./logs/wl-domain.log
  FileTimeSpan: 24
  Name: examples
  Notes:
  NumberOfFilesLimited: false
  ObjectName: examples
  Registered: false
  RotationTime: 00:00
  RotationType: none
  Type: LogConfig
```
CHAPTER

3 Accessing and Changing Configuration Information

WebLogic Server managed resources are configured from the values in Local Configuration MBeans, which are replicas of the Administration MBeans on the Administration Server.

If you want to programmatically view or change the configuration data for managed resources, you must first use the MBeanServer interface or the WebLogic Server type-safe interface to retrieve Local Configuration MBeans or Administration MBeans. Then you use APIs in the weblogic.management.configuration package to view or change the configuration data. For information about viewing the API documentation, refer to “Documentation for Configuration MBean APIs” on page 1-11.

Note: The values in the Local Configuration MBeans can differ from the Administration MBeans if you use a startup option to override the Administration MBean values, or if you use an API to change values in a Local Configuration MBean directly. For more information about the distribution of configuration data in a WebLogic Server domain, refer to “WebLogic Server Managed Resources and MBeans” on page 1-2.
This topic provides examples for programmatically retrieving and modifying the configuration of WebLogic Server resources using the `weblogic.Admin` utility, the JMX `MBeanServer` APIs, and the WebLogic Server type-safe interface:

- “Example: Using `weblogic/Admin` to Configure the Message Level for Standard Out” on page 3-3
- “Example: Using `MBeanServer` to Configure the Message Level for Standard Out” on page 3-4
- “Example: Using the Type-Safe Interface to Retrieve Information About a JMS Configuration” on page 3-6
Example: Using weblogic.Admin to Configure the Message Level for Standard Out

This example uses weblogic.Admin to find the WebLogicObjectName of the Local Configuration MBean instance of weblogic.management.configuration.ServerMBean. Then it sets the level of messages that a server instance named peach sends to standard out. Because it sets the value of a Local Configuration MBean, the updated value applies only to the current server session.

Listing 3-1 Configuring the Message Level

```
C:\myWLDomains\mydomain>java weblogic.Admin -url http://peach:8001 -username weblogic -password weblogic GET -pretty -type ServerConfig
---------------------------
MBeanName: "mydomain:Location=peach,Name=peach,Type=ServerConfig"
   AcceptBacklog: 50
   AdministrationPort: 0
   ...
   StdoutDebugEnabled: false
   StdoutEnabled: true
   StdoutFormat: standard
   StdoutLogStack: true
   StdoutSeverityLevel: 16
C:\myWLDomains\mydomain>java weblogic.Admin -url http://peach:8001 -username weblogic -password weblogic SET -mbean mydomain:Location=peach,Name=peach,Type=ServerConfig
   -property StdoutSeverityLevel 64
Ok
```

The weblogic.Admin utility returns the string Ok to indicate that the SET command succeeded.
Example: Using MBeanServer to Configure the Message Level for Standard Out

The class in this example uses the Local Configuration MBean instance of `weblogic.management.configuration.ServerMBean` to temporarily change the level of messages that a server instance named `peach` sends to standard out. It uses the standard JMX MBeanServer interface to change the `ServerMBean` configuration.

The class as written runs on the Administration Server and uses the Administration MBeanHome to retrieve the Local Configuration MBean for `peach`, but you could modify it to run on a local server and use the local server’s MBeanHome to retrieve `ServerMBean`.

Instead of retrieving a list of all MBeans and then filtering the list to find the local `ServerMBean` for a specific server instance, this example uses the MBean naming conventions to construct the `WebLogicObjectName` for the `ServerMBean`. For more information about naming conventions, refer to “WebLogicObjectNames for WebLogic Server MBeans” on page 2-16.

In the example, `weblogic` is a user who has permission to view attributes of the `ServerMBean`. For information about permissions to view and modify MBeans, refer to "Protecting System Administration Operations" in the WebLogic Server Administration Guide.

Listing 3-2 Configuring EJB Deployment Descriptors

```java
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.*;
import javax.management.MBeanServer;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.WebLogicObjectName;

public class changeStandardOut {
```

3-4 Programming WebLogic Management Services with JMX
Example: Using MBeanServer to Configure the Message Level for Standard Out

```java
public static void main(String[] args) {
    MBeanHome home = null;
    MBeanServer homeServer = null;

    // domain variables
    String url = "t3://localhost:7001";
    String username = "weblogic";
    String password = "weblogic";

    // setting the initial context
    try {
        Environment env = new Environment();
        env.setProviderUrl(url);
        env.setSecurityPrincipal(username);
        env.setSecurityCredentials(password);
        Context ctx = env.getInitialContext();
    } catch (Exception e) {
        System.out.println("Exception caught: " + e);
    }

    // getting the Administration MBeanHome
    home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
    } catch (Exception e) {
        System.out.println("Exception caught: " + e);
    }

    // Constructing a WebLogicObjectName for the local ServerMBean
    // that is associated with the server instance named peach.
    String mbeanName = "examples:Location=peach,Name=peach,Type=ServerConfig";
    WebLogicObjectName objName = new WebLogicObjectName(mbeanName);

    // Retrieving the MBeanServer interface.
    homeServer = home.getMBeanServer();

    // Using MBeanServer to set the value of the StdoutSeverityLevel attribute
    homeServer.setAttribute(mbeanName, StdoutSeverityLevel, 64);

    // Providing feedback that operation succeeded.
    System.out.println("Changed standard out severity level to: " +
                       homeServer.getAttribute(mbeanName, StdoutSeverityLevel));
    } catch (Exception e) {
        System.out.println("Caught exception: " + e);
    }
}
```
The example in this section collects all the JMS-related configuration information in a domain. To retrieve JMS-related Administration MBeans, it uses the `getMBeansByType` method of the Administration MBeanHome.

You could add more functionality to this program by using the get and set methods of the JMS-related Administration MBeans that the program retrieves.

```java
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.*;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import javax.management.ObjectName;
import weblogic.management.WebLogicMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.runtime.ServerRuntimeMBean;

public class getJMSInfo {
    public static void main(String[] args) {
        MBeanHome home = null;
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";
        WebLogicMBean bean = null;

        //setting the initial context
        try {
            Environment env = new Environment();
            // code...
Example: Using the Type-Safe Interface to Retrieve Information About a JMS Config-

```java
env.setProviderUrl(url);
env.setSecurityPrincipal(username);
env.setSecurityCredentials(password);
Context ctx = env.getInitialContext();
System.out.println("got the IC");

// getting the Administration MBeanHome
home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
System.out.println("Got the MBeanHome " + home);
} catch (Exception e) {
    System.out.println("Exception caught: " + e);
}

/* Getting the Administration MBeans. The getMBeansByType method
   * retrieves Administration MBeans because there is no "Config"
   * or "Runtime" suffix in the value that is passed.
   * For example, "JMSConnectionFactory" retrieves an Administration
   * MBean, while "JMSConnectionFactoryConfig" retrieves a Configuration
   * MBean.
   * Set myset = home.getMBeansByType("JMSConnectionFactory");
   * Iterator iter = myset.iterator();
   * System.out.println("Iterating over: ");
   * while(iter.hasNext()) {
   *    bean = (WebLogicMBean) iter.next();
   *    System.out.println("Got the server mbean: " + bean);
   * }
   *
   * myset = home.getMBeansByType("JMSConnectionFactory");
   * iter = myset.iterator();
   * System.out.println("Iterating over: ");
   * while(iter.hasNext()) {
   *    bean = (WebLogicMBean) iter.next();
   *    System.out.println("Got the config mbean: " + bean);
   * }
   *
   * myset = home.getMBeansByType("JMSConnectionPool");
   * iter = myset.iterator();
   * System.out.println("Iterating over: ");
   * while(iter.hasNext()) {
   *    bean = (WebLogicMBean) iter.next();
   *    System.out.println("Got the runtime mbean: " + bean);
   * }
   *
   * myset = home.getMBeansByType("JMSQueue");
   * iter = myset.iterator();
   * System.out.println("Iterating over: ");
   * while(iter.hasNext()) {
   *    bean = (WebLogicMBean) iter.next();
   *    System.out.println("Got the runtime mbean: " + bean);
   * }
   */
```
3  Accessing and Changing Configuration Information

myset = home.getMBeansByType("JMSConnectionFactory");
iter = myset.iterator();
System.out.println("Iterating over: ");
while(iter.hasNext()) {
    bean = (WebLogicMBean) iter.next();
    System.out.println("Got the runtime mbean: " + bean);
}

myset = home.getMBeansByType("JMSConnectionFactory");
iter = myset.iterator();
System.out.println("Iterating over: ");
while(iter.hasNext()) {
    bean = (WebLogicMBean) iter.next();
    System.out.println("Got the runtime mbean: " + bean);
}
CHAPTER

4 Using WebLogic Server MBean Notifications

To report changes in configuration and runtime information, all WebLogic Server MBeans emit JMX notifications. A notification is a JMX object that describes a state change or some other specific condition that has occurred in an underlying resource.

You can create Java classes called listeners that listen for these notifications. For example, your application can include a listener that receives notifications when applications are deployed, undeployed, or redeployed.

This topic includes the following sections:

- “WebLogic Server Notification Types” on page 4-2
- “Listening for Notifications: Main Steps” on page 4-3
WebLogic Server Notification Types

WebLogic Server MBeans implement the `javax.management.NotificationBroadcaster` interface to emit different types of notification objects depending on the type of event that occurs:

- When an MBean’s attribute value changes, it emits a `javax.management.AttributeChangeNotification` object.
- When an MBean’s `addAttributeName` method is called, it emits a `weblogic.management.AttributeAddNotification` object.
- When an MBean’s `removeAttributeName` method is called, it emits a `weblogic.management.AttributeRemoveNotification` object.

In addition, when MBeans have been registered or unregistered, the WebLogic Server JMX services emit notifications of type `javax.management.MBeanServerNotification`.

For more information about the `javax.management` notification types, refer to the JMX 1.0 API documentation, which you can download from [http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html](http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html). The archive that you download includes the API documentation.

For more information about the `weblogic.management` notification types, refer to the Javadoc for `AttributeAddNotification` and `AttributeRemoveNotification`.

WebLogic Server Log Notifications

When a WebLogic Server resource generates a log message, its MBeans emit a notification of type `weblogic.management.WebLogicLogNotification`. You can use the `WebLogicLogNotification` API to extract parts of the log message, including the transaction ID, user ID, and version number associated with the message.

For more information about log notifications, refer to the *Using WebLogic Logging Services* guide.
Listening for Notifications: Main Steps

To listen for the notifications that MBeans emit, do the following:

1. Create a listener class in your application.
2. Register the class with the MBeans whose notifications you want to receive.
3. Optionally implement and register a NotificationFilter class, which provides additional control over which notifications the listener receives.

Figure 4-1 shows a basic system in which a JSP contains a NotificationListener that is registered with an MBean’s implementation of the NotificationBroadcaster interface.

This section contains the following subsections:

- Creating a Notification Listener
- Registering a Notification Listener

For a complete explanation of JMX notifications and how they work, download the JMX 1.0 specification from http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html.
Creating a Notification Listener

To create a notification listener for a client that runs within the same JVM as WebLogic Server, create a class that implements javax.management.NotificationListener. Your implementation must include the NotificationListener.handleNotification() method.

For more information on NotificationListener, refer to the javax.management.Notification Javadoc in the JMX 1.0 API documentation, which you can download from http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html. The archive that you download includes the API documentation.

To create a notification listener for a client that runs in a JVM that is separate from WebLogic Server, create a class that implements weblogic.management.RemoteNotificationListener. RemoteNotificationListener extends javax.management.NotificationListener and java.rmi.Remote, making MBean notifications available to external clients via RMI. Your implementation must include the RemoteNotificationListener.handleNotification() method. For more information, refer to RemoteNotificationListener Javadoc.
After you implement `RemoteNotificationListener`, you register your listener with MBeans whose notifications you want to receive. (See Figure 4-2.)

**Figure 4-2  Monitoring Notifications from a Separate JVM**

The following example creates a remote listener that prints output when `NotificationBroadcaster` broadcasts a `WebLogicLogNotification` message that indicates an application has been deployed or undeployed.

**Listing 4-1  Notification Listener**

```java
import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.NotificationListener;
import javax.management.Notification;
import weblogic.management.RemoteNotificationListener;
import weblogic.management.logging.WebLogicLogNotification;

public class myListener implements RemoteNotificationListener {
    public void handleNotification(Notification notification, Object obj) {
        WebLogicLogNotification wln = (WebLogicLogNotification)notification;
    }
}
```
Using WebLogic Server MBean Notifications

/*
  * These are all the attributes you can get on a
  * WebLogicLogNotification
  */

System.out.println("message id = " + wln.getMessageId());
System.out.println("server name = " + wln.getServername());
System.out.println("machine name = " + wln.getMachineName());
System.out.println("severity = " + wln.getSeverity());
System.out.println("type = " + wln.getType());
System.out.println("timestamp = " + wln.getTimeStamp());
System.out.println("message = " + wln.getMessage());
System.out.println("thread id = " + wln.getThreadid());
System.out.println("user id = " + wln.getUserId());
System.out.println("transaction id = " + wln.getTransactionId());
System.out.println("version = " + wln.getVersion());

int messageId = wln.getMessageId();

/* These are the messageIDs of the messages broadcast when an
  * application is deployed/undeployed/redeployed
  * 160004 is for undeployment
  * 160003 is for deployment
  */

if(messageId == 160004)
  System.out.println("wln.getMessage();
else if (messageId == 160003)
  System.out.println(wln.getMessage());
else;
}
}

Registering a Notification Listener

Because all WebLogic Server MBeans implement the
javax.management.NotificationBroadcaster interface, you can register a
NotificationListener with any MBean.

Registering a NotificationListener can be accomplished by calling the MBean’s
addNotificationListener() method. However, in most cases it is preferable to
use the addNotificationListener() method of the MBeanServer interface, which
saves the trouble of looking up a particular MBean simply for registration purposes.
The following example uses `MBeanServer.addNotificationListener()` to register the listener from Listing 4-1 with the `LogBroadcasterRuntimeMBean`.

**Listing 4-2 Registering a Listener**

```java
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.*;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import javax.management.ObjectName;
import weblogic.management.WebLogicMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.runtime.LogBroadcasterRuntimeMBean;
import weblogic.management.WebLogicObjectName;
import javax.management.*;
import javax.management.Notification;
import weblogic.management.RemoteMBeanServer;
/* This class is to be registered as a startup class with the server that
* receives the Log Notifications
*/
public class logger {
    public static void main(String[] args) {
        MBeanHome home = null;
        LogBroadcasterRuntimeMBean logBroadcaster = null;
        RemoteMBeanServer rmbs = null;
        //domain variables
        String serverName = "MyServer";
        String domainName = "myDomain";
        try {
            Context ctx = new InitialContext();
            //Get a local MBeanHome
            home = (MBeanHome) ctx.lookup("weblogic.management.home." + serverName);
            } catch (Exception e) {
                System.out.println("Exception caught: " + e);
            }
            //Use MBeanHome to get MBeanServer
            try {
```
Using WebLogic Server MBean Notifications

```java
rmbs = home.getMBeanServer();
} catch(Exception e) {
    System.out.println("Caught exception: " + e);
}

try {
    /* The LogBroadcasterRuntimeMBean is only responsible for emitting
       notifications for log messages. All notifications generated are
       of the type WebLogicLogNotification. There is only one
       LogBroadcasterRuntimeMBean per server.
    */
    WebLogicObjectName oname = new WebLogicObjectName(domainName + "
        :Name=TheLogBroadcaster,Type=LogBroadcasterRuntime,Location="
        + serverName);
    myListener listener = new myListener();

    rmbs.addNotificationListener(oname, listener, null, null);
    System.out.println("\n[myListener]: Listener registered
    for the LogBroadcasterRuntimeMBean ...");
} catch(Exception e) {
    System.out.println("Exception: " + e);
}
```
CHAPTER

5 Accessing Runtime Information

WebLogic Server includes a large number of MBeans that provide information about the runtime state of managed resources. If you want to create applications that view and modify this runtime data, you must first use the MBeanServer interface or the WebLogic Server type-safe interface to retrieve Runtime MBeans. Then you use APIs in the weblogic.management.runtime package to view or change the runtime data. For information about viewing the API documentation, refer to “Documentation for Runtime MBean APIs” on page 1-14.

This topic provides examples for retrieving and modifying runtime information about WebLogic Server domains and server instances:

- “Determining the Active Domain and Servers” on page 5-2
- “Example: Viewing and Changing the Runtime State of a WebLogic Server Instance” on page 5-5
- “Example: Viewing Runtime Information About Clusters” on page 5-16
Determining the Active Domain and Servers

The Administration MBeanHome interface includes APIs that you can use to determine the name of the currently active domain and the name of all server instances that are currently active.

The example class in Listing 5-1 does the following:

1. Retrieves the Administration MBeanHome interface.
2. Uses MBeanHome.getActiveDomain().getName() to retrieve the name of the domain.
3. Uses the getMBeansByType method to retrieve the set of all ServerRuntime MBeans in the domain.
4. Iterates through the set and compares the names of the ServerRuntimeMBean instances with the name of the WebLogic Server instance. If the instance is active, it prints the name of the server.

In the following example, weblogic is the username and password for a user who has permission to view and modify MBean attributes. For information about permissions to modify MBeans, refer to "Protecting System Administration Operations" in the WebLogic Server Administration Guide.

The code in this example must run on the Administration Server.

Listing 5-1  Determining the Active Domain and Servers

```java
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.*;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import javax.management.ObjectName;
import weblogic.management.WebLogicMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.runtime.ServerRuntimeMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.management.MBeanHome;
```
public class getActiveDomainAndServers {
    public static void main(String[] args) {
        MBeanHome home = null;

        // url of the Administration Server
        String url = "t3://localhost:7001";
        String username = "weblogic";
        String password = "weblogic";
        ServerRuntimeMBean serverRuntime = null;
        int count = 0;

        Set mbeanSet = null;
        Iterator mbeanIterator = null;

        // setting the initial context
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();
            System.out.println("got the IC");

            // getting the Administration MBeanHome
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
        } catch (Exception e) {
            System.out.println("Exception caught: " + e);
        }

        // getting the name of the active domain
        try {
            System.out.println("Active Domain: " + home.getActiveDomain().getName());
        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }

        // getting the names of servers in the domain
        System.out.println("Active Servers: ");
        mbeanSet = home.getMBeansByType("ServerRuntime");
        mbeanIterator = mbeanSet.iterator();
        while(mbeanIterator.hasNext()) {
            serverRuntime = (ServerRuntimeMBean)mbeanIterator.next();
        }
    }
}
5  Accessing Runtime Information

//printing the names of active servers
if(serverRuntime.getState().equals("Running")) {
    System.out.println("Name: " + serverRuntime.getName());
    System.out.println("ListenAddress: " + serverRuntime.getListenAddress());
    System.out.println("ListenPort: " + serverRuntime.getListenPort());
    count++;
}
System.out.println("Number of servers active in the domain: " + count);
}

Using weblogic.Admin to Determine Active Domains and Servers

While you can use the example code in Listing 5-1 to determine active domains and servers from a JMX application, you can use the weblogic.Admin utility to accomplish a similar task from the command line or a script.

The following command returns the name of the currently active domain, where peach hosts the domain’s Administration Server and weblogic is the name and password of a user who has permission to view MBean attributes:

```
java weblogic.Admin -url peach:8001 -username weblogic -password weblogic GET -type DomainRuntime -property Name
```

The command output includes the WebLogicObjectName of the DomainRuntimeMBean and the value of its Name attribute:

```
{MBeanName="examplesDomain:Location=peach,Name=examplesDomain,ServerRuntime=peach,Type=DomainRuntime"{Name=examplesDomain}}
```
Example: Viewing and Changing the Runtime State of a WebLogic Server Instance

To see a list of all server instances that are currently active, you use ask the Administration Server to retrieve all ServerRuntime MBeans that are registered in its Administration MBeanHome interface. (Only active server instances register ServerRuntime MBeans with the Administration MBeanHome interface.)

You must specify the -adminurl argument to instruct the GET command to use the Administration Server’s Administration MBeanHome interface:

```
java weblogic.Admin -adminurl peach:8001 -username weblogic -password weblogic GET -type ServerRuntime -property State
```

The command output includes the WebLogicObjectName of all ServerRuntime MBeans and the value of each State attribute:

```
------------------------------
MBeanName:
"MedRec:Location=MedRecMS2,Name=MedRecMS2,Type=ServerRuntime"
  State: RUNNING
------------------------------
MBeanName:
"MedRec:Location=MedRecServer,Name=MedRecServer,Type=ServerRuntime"
  State: RUNNING
------------------------------
MBeanName:
"MedRec:Location=MedRecMS1,Name=MedRecMS1,Type=ServerRuntime"
  State: RUNNING
```

Example: Viewing and Changing the Runtime State of a WebLogic Server Instance

The weblogic.management.runtime.ServerRuntimeMBean interface provides runtime information about a WebLogic Server instance. For example, it indicates which listen ports and addresses a server is using. It also includes operations that change the lifecycle state of a server. (For information about server states, refer to "Server Lifecycle" in the Configuring and Managing WebLogic Server guide.)
This section provides examples of finding `ServerRuntimeMBean` and using it to change the state of a server instance. Each example illustrates a different way of retrieving `ServerRuntimeMBean`:

- “Using a Local MBeanHome and `getRuntimeMBean()`” on page 5-6
- “Using the Administration MBeanHome and `getMBean()`” on page 5-11
- “Using the Administration MBeanHome and `getMBeansByType()`” on page 5-9
- “Using the `MBeanServer` Interface” on page 5-14

You cannot use the `weblogic.Admin` utility to change the value of Runtime MBean attributes.

## Using a Local MBeanHome and `getRuntimeMBean()`

Each WebLogic Server instance hosts its own `MBeanHome` interface, which provides access to the Local Configuration and Runtime MBeans on the server instance. As opposed to using the Administration `MBeanHome` interface, using the local `MBeanHome` saves you the trouble of filtering MBeans to find those that apply to the current server. It also uses fewer network hops to access MBeans, because you are connecting directly to the server (instead of routing requests through the Administration Server).

The `MBeanHome` interface includes the `getRuntimeMBean()` method, which returns only Runtime MBeans that reside on the current WebLogic Server. If you invoke `MBeanHome.getRuntimeMBean()` on the Administration Server, it returns only the Runtime MBeans that are on the Administration Server.

In the following example, `weblogic` is the username and password for a user who has permission to view and modify MBean attributes and `Server1` is the name of the WebLogic Server instance for which you want to view and change status. For information about permissions to modify MBeans, refer to "Protecting System Administration Operations" in the *WebLogic Server Administration Guide*.

### Listing 5-2 Using a Local MBeanHome and `getRuntimeMBean()`

```java
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
```
import javax.naming.*;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import javax.management.ObjectName;
import weblogic.management.WebLogicMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.runtime.ServerRuntimeMBean;
import weblogic.management.WebLogicObjectName;

public class serverRuntimeInfo1 {
    public static void main(String[] args) {
        MBeanHome home = null;

        //domain variables
        String url = "t3://localhost:7003";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";

        ServerRuntimeMBean serverRuntime = null;
        ServerRuntimeMBean serverRuntimeM = null;

        //setting the initial context
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            //getting the local MBeanHome
            home = (MBeanHome) ctx.lookup("weblogic.management.home." + serverName);
            System.out.println("Got the MBeanHome: " + home + " for server: " + serverName);
        } catch (Exception e) {
            System.out.println("Exception caught: " + e);
        }

        /* Here we use the getRuntimeMBean method to access the ServerRuntimeMBean
        * of the server instance.
        */

        try {
            serverRuntime =
                    (ServerRuntimeMBean) home.getRuntimeMBean(serverName, "ServerRuntime");
            System.out.println("Got serverRuntimeMBean: " + serverRuntime);
        } catch (javax.management.InstanceNotFoundException e) {
            
    
}
System.out.println("Caught exception: "+ e);
}
System.out.println("Current state: "+ serverRuntime.getState());
System.out.println("Suspending the server ...");
serverRuntime.suspend();
System.out.println("Current state: "+ serverRuntime.getState());
System.out.println("Stopping the server ...");

//changing the state to SHUTDOWN
serverRuntime.shutdown();
System.out.println("Current state: "+ serverRuntime.getState());
}
Using the Administration MBeanHome and getMBeansByType()

Like the example in Listing 5-1, “Determining the Active Domain and Servers,” on page 5-2, the example class in this section uses the Administration MBeanHome interface to retrieve a ServerRuntime MBean. The Administration MBeanHome provides a single access point for all MBeans in the domain, but it requires you to either construct the WebLogicObjectName of the MBean you want to retrieve or to filter MBeans to find those that apply to a specific current server.

This example class uses MBeanHome.getMBeansByType method to retrieve the set of all ServerRuntime MBeans in the domain. It then iterates through the set and compares the names of the ServerRuntimeMBean instances with the name of a WebLogic Server instance. When it finds a specific server instance, the class changes the state of the server to SHUTDOWN.

In the following example, weblogic is the username and password for a user who has permission to view and modify MBean attributes, Server1 is the name of the WebLogic Server instance for which you want to view and change status, and mihirDomain is the name of the WebLogic Server administration domain.

For information about permissions to modify MBeans, refer to “Protecting System Administration Operations” in the WebLogic Server Administration Guide.

Listing 5-3 Using the Administration MBeanHome and getMBeansByType()

```java
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.*;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import javax.management.ObjectName;
import weblogic.management.WebLogicMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.runtime.ServerRuntimeMBean;
import weblogic.management.WebLogicObjectName;
```
public class serverRuntimeInfo3 {
    public static void main(String[] args) {
        MBeanHome home = null;

        // domain variables
        String url = "t3://localhost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";

        ServerRuntimeMBean serverRuntime = null;
        Set mbeanSet = null;
        Iterator mbeanIterator = null;

        // Setting the initial context
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            // Getting the Administration MBeanHome.
            home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
            System.out.println("Got the Admin MBeanHome: " + home);
        } catch (Exception e) {
            System.out.println("Exception caught: " + e);
        }

        /* Here we use the getMBeansByType method to get the set of ServerRuntime mbeans
         * Then we iterate through the set. We retrieve the ServerRuntimeMBean we are
         * interested in by comparing the name to the value of serverName.
         */
        try {
            mbeanSet = home.getMBeansByType("ServerRuntime");
            mbeanIterator = mbeanSet.iterator();
            while (mbeanIterator.hasNext()) {
                serverRuntime = (ServerRuntimeMBean) mbeanIterator.next();
                if (serverRuntime.getName().equals(serverName)) {
                    System.out.println("we have got the serverRuntimeMembean: " + serverRuntime + 
                        " for: " + serverName);
                    System.out.println("Current state: " + serverRuntime.getState());
                    System.out.println("Suspending the server ...");  
                    System.out.println("Stopping the server ...");
                }
            }
        } catch (Exception e) {
            System.out.println("Exception caught: " + e);
        }
    }
}

5 Accessing Runtime Information

5-10 Programming WebLogic Management Services with JMX
Example: Viewing and Changing the Runtime State of a WebLogic Server Instance

//changing the state to SHUTDOWN
serverRuntime.shutdown();
System.out.println("Current state: " + serverRuntime.getState());
} catch (javax.management.InstanceNotFoundException e) {
    System.out.println("Caught exception: " + e);
}

Using the Administration MBeanHome and getMBean()

Instead of retrieving a list of all MBeans and then filtering the list to find the ServerRuntimeMBean for a specific server, this example uses the MBean naming conventions to construct the WebLogicObjectName for the ServerRuntimeMBean on a server instance named Server1. For information about constructing a WebLogicObjectName, refer to "WebLogicObjectNames for WebLogic Server MBeans" on page 2-16.

To make sure that you supply the correct object name, you can use the weblogic.Admin GET command. For example, the following command returns the object name and list of attributes of the ServerRuntimeMBean for a server instance named Server1:


For more information about using the weblogic/Admin utility to find information about MBeans, refer to "MBean Management Command Reference" in the WebLogic Server Command Line Reference.
In Listing 5-4, weblogic is the username and password for a user who has permission to view and modify MBean attributes, Server1 is the name of the WebLogic Server instance for which you want to view and change status, and mihirDomain is the name of the WebLogic Server administration domain.

**Listing 5-4 Using the Administration MBeanHome and getMBean()**

```java
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.*;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import javax.management.ObjectName;
import weblogic.management.WebLogicMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.runtime.ServerRuntimeMBean;
import weblogic.management.WebLogicObjectName;

public class serverRuntimeInfo2 {
    public static void main(String[] args) {
        //domain variables
        String url = "t3://localhost:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";
        ServerRuntimeMBean serverRuntime = null;

        //setting the initial context
        try {
            Environment env = new Environment();
            env.setProviderUrl(url);
            env.setSecurityPrincipal(username);
            env.setSecurityCredentials(password);
            Context ctx = env.getInitialContext();

            /* Getting the Administration MBeanHome. */
            /* Note: Looking up MBeanHome.ADMIN_JNDI_NAME returns the Administration */
            /* MBeanHome interface. It provides access to all MBeans in the domain. */
            /* Looking up "weblogic.management.home.<AdminServerName>" returns the */
            /* local MBeanHome for the Administration Server. It provides */
        }
    }
}
```
Example: Viewing and Changing the Runtime State of a WebLogic Server Instance

/*
 to the Configuration and Runtime MBeans on the Administration Server.
*/
home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
System.out.println("Got the Admin MBeanHome: " + home + " from the
Admin server");
} catch (Exception e) {
    System.out.println("Exception caught: " + e);
}

try {
    /* Creating the mbean object name.
    * The serverName refers to the name of the Managed Server that hosts
    * the ServerRuntimeMBean.
    */
    String name = "mihirDomain:Location=" + serverName + ",Name=" +
    serverName + ",Type=ServerRuntime";
    WebLogicObjectName objName = new WebLogicObjectName(name);
    System.out.println("Created WebLogicObjectName: " + name);

    serverRuntime = (ServerRuntimeMBean)home.getMBean(objName);
    System.out.println("Got the serverRuntime using the adminHome: " +
    serverRuntime");
} catch(Exception e) {
    System.out.println("Exception: " + e);
}

    System.out.println("Current state: " + serverRuntime.getState() );
    System.out.println("Suspending the server ...");
    serverRuntime.suspend();
    System.out.println("Current state: " + serverRuntime.getState() );
    System.out.println("Stopping the server ...");

    //changing the state to SHUTDOWN
    serverRuntime.shutdown();
    System.out.println("Current state: " + serverRuntime.getState() );

}
Using the MBeanServer Interface

The example in this section uses a standard JMX approach for interacting with MBeans. It uses the Administration MBeanHome interface to retrieve the javax.management.MBeanServer interface and then uses MBeanServer to retrieve the value of the ListenPort attribute of the ServerRuntimeMBean for a server instance named Server1.

In the following example, weblogic is the username and password for a user who has permission to view and modify MBean attributes and mihirDomain is the name of the WebLogic Server administration domain.

Listing 5-5 Using the Administration MBeanHome and getMBean()

```java
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.
import javax.management.ObjectName;
import javax.management.MBeanServer;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.runtime.ServerRuntimeMBean;
import weblogic.management.WebLogicObjectName;

public class serverRuntimeInfo3 {
    public static void main(String[] args) {
        MBeanHome home = null;
        //domain variables
        String url = "t3://adminserver:7001";
        String serverName = "Server1";
        String username = "weblogic";
        String password = "weblogic";
        String ListenPort = "7001";
        ServerRuntimeMBean serverRuntime = null;
        //setting the initial context
        try {
```
Example: Viewing and Changing the Runtime State of a WebLogic Server Instance

Environment env = new Environment();
env.setProviderUrl(url);
env.setSecurityPrincipal(username);
env.setSecurityCredentials(password);
Context ctx = env.getInitialContext();

// Getting the Administration MBeanHome.
home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
System.out.println("Got the Admin MBeanHome: " + home + " from the Admin server");
} catch (Exception e) {
    System.out.println("Exception caught: " + e);
}

try {
    /* Creating the mbean object name.
     * The serverName refers to the name of the Managed Server that hosts
     * the ServerRuntimeMBean.
     */
    String name = "mihirDomain:Location=", serverName + ",Name=", serverName + ",Type=ServerRuntime";
    WebLogicObjectName objName = new WebLogicObjectName(name);
    System.out.println("Created WebLogicObjectName: " + name);

    //Retrieving the MBeanServer interface
    homeServer = home.getMBeanServer();

    //Retrieving the ListenPort attribute of  ServerRuntimeMBean
    attributeValue = homeServer.getAttribute(objName, ListenPort);
    System.out.println("ListenPort for " + serverName + " is:" + attributeValue);
} catch(Exception e) {
    System.out.println("Exception: " + e);
}
Example: Viewing Runtime Information About Clusters

The example in this section retrieves the number and names of WebLogic Server instances currently running in a cluster. It uses `weblogic.management.runtime.ClusterRuntimeMBean`, which provides information about a single Managed Server’s view of the members of a WebLogic cluster.

Only Managed Servers host instances of `ClusterRuntimeMBean`, and you must retrieve the `ClusterRuntimeMBean` instance from a Managed Server that is actively participating in a cluster.

To make sure that it retrieves a `ClusterRuntimeMBean` from an active Managed Server that is in a cluster, this example does the following:

1. Retrieves the Administration MBeanHome, which runs on the Administration Server and can provide access to all `ClusterRuntimeMBeans` in the domain.
2. Retrieves all `ClusterRuntimeMBeans` and determines whether they belong to a specific domain.
3. Finds one `ClusterRuntimeMBean` for a Managed Server in the domain of interest.
4. Uses the `ClusterRuntimeMBean` APIs on the Managed Server to determine the number and name of active servers in the cluster.
Example: Viewing Runtime Information About Clusters

In the example, weblogic is the username and password for a user who has permission to view and modify MBean attributes. For information about permissions to modify MBeans, refer to "Protecting System Administration Operations" in the WebLogic Server Administration Guide.

Listing 5-6 Retrieving a List of Servers Running in a Cluster

```java
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.*;
import weblogic.jndi.Environment;
import weblogic.jndi.MBeanHome;
import java.rmi.RemoteException;
import javax.jndi.NamingEnumeration;
import javax.management.ObjectName;
import weblogic.management.MBeanHome;
import weblogic.management.WebLogicMBean;
import weblogic.management.runtime.ClusterRuntimeMBean;
import weblogic.management.WebLogicObjectName;
import weblogic.management.MBeanHome;

public class getRunningServersInCluster {
    public static void main(String[] args) {
        MBeanHome home = null;

        //domain variables
        String url = "t3://localhost:7001"; //url of the Administration Server

        /* If you have more than one cluster in your domain, define a list of all the
         * servers in the cluster. You compare the servers in the domain with this list
         * to determine which servers are in a specific cluster.
         */

        String server1 = "cs1"; // name of server in the cluster
        String server2 = "cs2"; // name of server in the cluster

        String username = "weblogic";
        String password = "weblogic";

        ClusterRuntimeMBean clusterRuntime = null;
        Set mbeanSet = null;
        Iterator mbeanIterator = null;
        String name = "";
        String[] aliveServerArray = null;
```
// Setting the initial context
try {
    Environment env = new Environment();
    env.setProviderUrl(url);
    env.setSecurityPrincipal(username);
    env.setSecurityCredentials(password);
    Context ctx = env.getInitialContext();

    // Getting the Administration MBeanHome.
    home = (MBeanHome) ctx.lookup(MBeanHome.ADMIN_JNDI_NAME);
    } catch (Exception e) {
        System.out.println("Exception caught: "+ e);
    }

    // Retrieving a list of ClusterRuntime MBeans in the domain.
    try {
        mbeanSet = home.getMBeansByType("ClusterRuntime");
        mbeanIterator = mbeanSet.iterator();
        while(mbeanIterator.hasNext()) {
            // Retrieving one ClusterRuntime MBean from the list.
            clusterRuntime = (ClusterRuntimeMBean)mbeanIterator.next();

            // Getting the name of the ClusterRuntime MBean.
            name = clusterRuntime.getName();

            // Determining if the current ClusterRuntimeMBean belongs to a
            // server in the cluster of interest.
            if(name.equals(server1) || name.equals(server2)) {
                // Using the current ClusterRuntimeMBean to retrieve the number of
                // servers in the cluster.
                System.out.println("\nNumber of active servers in the cluster: "+
                        clusterRuntime.getAliveServerCount());

                // Retrieving the names of servers in the cluster.
                aliveServerArray = clusterRuntime.getServerNames();
                break;
            }
        }
    } catch (Exception e) {
        System.out.println("Caught exception: "+ e);
    }

    if(aliveServerArray == null) {
        System.out.println("\nThere are no running servers in the cluster");
        System.exit(1);
    }

5-18    Programming WebLogic Management Services with JMX
Example: Viewing Runtime Information About Clusters

```
System.out.println("\nThe running servers in the cluster are: ");
for (int i=0; i < aliveServerArray.length; i++) {
    System.out.println("server " + i + " : " + aliveServerArray[i]);
}
```


5 Accessing Runtime Information
CHAPTER 6

Monitoring WebLogic Server MBeans

WebLogic Server includes a set of monitor MBeans that emit JMX notifications only when specific MBean attributes change beyond a specific threshold. A monitor MBean observes the attribute of another MBean (the observed MBean) at specific intervals. The monitor derives a value from this observation, called the derived gauge. This derived gauge is either the exact value of the observed attribute, or optionally, the difference between two consecutive observed values of a numeric attribute.

When the value of the derived gauge satisfies a set of conditions, the monitor MBean emits a specific notification type. Monitors can also send notifications when certain error cases are encountered while monitoring an attribute value.

The process for monitoring an attribute of an MBean is as follows:

1. Create a listener class that can listen for notifications from monitor MBeans.
2. Choose a monitor MBean type that matches the type of data you want to observe.
3. Configure and instantiate a monitor MBean.
4. Instantiate the listener class.

This topic contains the following sections:

- “Creating a Notification Listener” on page 6-2
- “Choosing a WebLogic Server Monitor Type” on page 6-3
- “Instantiating the Monitor and Listener” on page 6-5
- “Notification Types” on page 6-13
- “Sample Monitoring Scenarios” on page 6-14
Creating a Notification Listener

As any other MBean, monitor MBeans emit notifications by implementing `javax.management.NotificationBroadcaster`. To create a listener for notifications from a monitor MBean, create a class that does the following:

1. Implements `NotificationBroadcaster` or `weblogic.management.RemoteNotificationListener`.
2. Includes the `NotificationListener.handleNotification()` or the `RemoteNotificationListener.handleNotification()` method.

You can register the same notification listener with instances of `LogBroadcasterMBean`, `monitor MBeans`, or any other MBean.

The example below creates a listener object for an application that runs in a JVM outside the WebLogic Server JVM. It includes logic that outputs additional messages when it receives notifications from monitor MBeans.

**Listing 6-1  Listener for Monitor Notifications**

```java
import java.rmi.Remote;
import javax.management.Notification;
import javax.management.monitor.MonitorNotification;
import weblogic.management.RemoteNotificationListener;

public class CounterListener
    implements RemoteNotificationListener {

    public void handleNotification(Notification notification, Object obj) {
        System.out.println("\n\n Notification Received ... ");
        System.out.println("Type= " + notification.getType());
        System.out.println("SequenceNumber= " + notification.getSequenceNumber());
        System.out.println("Source= " + notification.getSource());
        System.out.println("Timestamp= " + notification.getTimeStamp() + "\n");
        if(notification instanceof MonitorNotification) {
            MonitorNotification monitorNotification = (MonitorNotification) notification;
            System.out.println("This notification is a MonitorNotification");
            System.out.println("Observed Attribute: " +
                                monitorNotification.getObservedAttribute());
        }
    }
```
System.out.println("Observed Object: " +
    monitorNotification.getObservedObject() );
System.out.println("Trigger value: " + monitorNotification.getTrigger() );
}
} }

Choosing a WebLogic Server Monitor Type

Monitor MBeans are specialized to observe changes in specific data types. You must instantiate the type of monitor MBean that matches the type of the object that an MBean returns for an attribute value. For example, a monitor MBean based on the StringMonitor type can observe an attribute that is declared as an Object as long as actual values of the attributes are String instances, as determined by the instanceof operator.

To choose a monitor type, do the following:

1. Determine the type of object that is returned by the MBean attribute that you want to observe by doing any of the following:
   - Refer to the WebLogic Server Javadoc.
   - Use the weblogic.Admin GET command, which provides information about the MBean that you specify. For more information, refer to "MBean Management Command Reference" in the WebLogic Server Administration Guide.
   - Use the javap command on the MBean you are monitoring. The javap command is a standard Java utility that disassembles a class file.

2. Choose a monitor type from the following table.

<table>
<thead>
<tr>
<th>A Monitor MBean of This Type</th>
<th>Observes This Object Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CounterMonitor</td>
<td>Integer</td>
</tr>
</tbody>
</table>
For more information about monitor types, refer to the JMX 1.0 specification, which you can download from [http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html](http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html). The archive that you download includes the API documentation.

<table>
<thead>
<tr>
<th>A Monitor MBean of This Type</th>
<th>Observes This Object Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>GaugeMonitor</td>
<td>Integer or floating-point (Byte, Integer, Short, Long, Float, Double)</td>
</tr>
<tr>
<td>StringMonitor</td>
<td>String</td>
</tr>
</tbody>
</table>
Instantiating the Monitor and Listener

After you determine which type of monitor to instantiate, you create a class that instantiates and configures the monitor, and instantiates and registers the listener.

This section contains the following subsections:
- “Main Steps for Instantiating a Monitor and Listener” on page 6-5
- “Example: Instantiating a CounterMonitor for a Remote Application” on page 6-7
- “Configuring CounterMonitor Objects” on page 6-9
- “Configuring GaugeMonitor Objects” on page 6-11
- “Configuring StringMonitor Objects” on page 6-12

Main Steps for Instantiating a Monitor and Listener

To instantiate a monitor MBean and listener object, create a class that does the following:

1. Creates a monitor object. The example in Listing 6-2 creates a `CounterMonitor` object using the default constructor of `javax.management.monitor.CounterMonitor`.

2. Configures the monitor object by doing the following:
   a. Constructs a JMX object name for the **monitor object**. Listing 6-2 uses `WebLogicObjectName()`, but you can use `javax.managementObjectName` for the monitor object. The object name must be unique throughout the entire WebLogic Server domain, and it must follow the JMX conventions:

   ```
   domain name:Name=name,Type=type[,attr=value]...
   ```

   b. Constructs a JMX object name for the **observed MBean** using `WebLogicObjectName()`.

   If the observed MBean is a WebLogic Server MBean, you must use `WebLogicObjectName()` instead of `javax.managementObjectName`. You...
can also use MBeanHome.getMBeansByType() or other WebLogic Server APIs to get the name of the observed MBean object. For examples of different methods of retrieving MBeans, refer to “Accessing WebLogic Server MBeans” on page 2-1.

c. Sets values for the monitor’s threshold parameters. The set of available parameters varies, depending on whether you are instantiating a CounterMonitor, GaugeMonitor, or StringMonitor.

For more information about the parameters that you pass to configure a monitor argument, refer to “Configuring CounterMonitor Objects” on page 6-9, “Configuring GaugeMonitor Objects” on page 6-11, and “Configuring StringMonitor Objects” on page 6-12.

d. Configures the monitor object using the monitor’s APIs.

3. Instantiates the listener object that you created in “Creating a Notification Listener” on page 6-2.

4. Registers the listener object using the monitor’s addNotificationListener() method.

5. (This step is needed only if your monitor class runs in a JVM that is outside the WebLogic Server JVM.) Initializes a reference to the monitor object within the MBean Server by doing the following:

a. Retrieving the MBeanServer interface using the Administration MBeanHome interface.

b. Using the monitor’s preRegister() method

6. Starts the monitor using the monitor’s start() method.
Example: Instantiating a CounterMonitor for a Remote Application

The following example creates a monitor for the ServicedRequestTotalCount attribute of the ExecuteQueueRuntimeMBean, which returns the number (int) of requests that have been processed by the corresponding execution queue.

Listing 6-2 Instantiating the Monitor and Listener

```java
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.*;
import weblogic.jndi.Environment;
import weblogic.management.MBeanHome;
import javax.naming.*;
import weblogic.management.WebLogicMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.*;
import weblogic.management.monitor.CounterMonitor;
import javax.management.*;
import javax.management.monitor.CounterMonitor;

public class clientMonitor {
    public static void main (String Args[]) {
        try {
            // Instantiate a CounterMonitor
            CounterMonitor monitor = new CounterMonitor();
            // construct the objectName for your CounterMonitor object
            WebLogicObjectName monitorObjectName = new WebLogicObjectName("mydomain:Type=CounterMonitor,Name=MyCounter");
            // Construct the objectName for the observed MBean
            WebLogicObjectName qObjectName = new WebLogicObjectName("mihirDomain:Location=MyServer,Name=default,
                ServerRuntime=MyServer,Type=ExecuteQueueRuntime");
            // Define variables to be used when configuring your CounterMonitor object.
            Integer threshold = new Integer(1000);
            Integer offset = new Integer(2000);
        }
    }
}
```
Monitoring WebLogic Server MBeans

//Configure your monitor object using the CounterMonitor APIs
  monitor.setThreshold(threshold);
  monitor.setNotify(true);
  monitor.setOffset(offset);
  monitor.setObservedObject(qObjectName);
  monitor.setObservedAttribute("ServicedRequestTotalCount");

//Instantiate and register your listener with the monitor
  CounterListener listener = new CounterListener();
  monitor.addNotificationListener(listener, null, null);

//Use the Administration MBeanHome API to get the MBeanServer interface.
//this is needed when you are registering a monitor from the
//client side.
  String url = "t3://localhost:7001"; //URL of the Administration server
  String username = "weblogic";
  String password = "weblogic";

  MBeanHome home = null;

  Environment env = new Environment();
  env.setProviderUrl(url);
  env.setSecurityPrincipal(username);
  env.setSecurityCredentials(password);
  Context ctx = env.getInitialContext();
  home = (MBeanHome) ctx.lookup(weblogic.management.adminhome);
  RemoteMBeanServer rmbs = home.getMBeanServer();

  monitor.preRegister(rmbs, monitorObjectName);

  //start the monitor
  monitor.start();

}  
catch (Exception e) {  e.printStackTrace();  }
}
Configuring CounterMonitor Objects

CounterMonitor objects observe changes in MBean attributes that are expressed as integers. The following list describes groups of CounterMonitor operations that you use to achieve typical configurations of a CounterMonitor instance:

- Sends a notification when the observed attribute exceeds the threshold.
  ```java
  setThreshold(int threshold);
  setNotify(true);
  setObservedObject(ObjectName);
  setObservedAttribute("AttributeName");
  ```

- Sends a notification when the observed attribute exceeds the threshold. Then it increases the threshold by the offset value. Each time the observed attribute exceeds the new threshold, the threshold is increased by the offset value. For example, if you set Threshold to 1000 and Offset to 2000, when the observed attribute exceeds 1000, the CounterMonitor object sends a notification and increases the threshold to 3000. When the observed attribute exceeds 3000, the CounterMonitor object sends a notification and increases the threshold again to 5000.
  ```java
  setThreshold(int threshold);
  setNotify(true);
  setOffset(int offset);
  setObservedObject(ObjectName);
  setObservedAttribute("AttributeName");
  ```

- Sends a notification when the observed attribute exceeds the threshold, and increases the threshold by the offset value. When the threshold reaches the value specified by the modulus, the threshold is returned to the value that was specified through the latest call to the monitor’s setThreshold method, before any offsets were applied. For example, if the original Threshold is set to 1000 and the Modulus is set to 5000, when the Threshold exceeds 5000, the monitor sends a notification and resets the Threshold to 1000.
  ```java
  setThreshold(int threshold);
  setNotify(true);
  setOffset(int offset);
  setModulus(int modulus);
  setObservedObject(ObjectName);
  setObservedAttribute("AttributeName");
  ```
Sends a notification when the difference between two consecutive observations exceeds the threshold. For example, the Threshold is 20 and the monitor observes an attribute value of 2. If the next observation is greater than 22, then the monitor sends a notification. However, if the value is 10 at the next observation, and 25 at the following observation, then the monitor does not send a notification because the value has not changed by more than 20 for any two consecutive observations.

```java
setThreshold(int threshold);
setNotify(true);
setDifferenceMode(true);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

Sends a notification when the difference between two consecutive observations exceeds the threshold, and increases the threshold by the offset value. When the threshold reaches the value specified by the modulus, the threshold is returned to the value that was specified through the latest call to the monitor's `setThreshold` method, before any offsets were applied.

```java
setThreshold(int threshold);
setNotify(true);
setOffset(int offset);
setModulus(int modulus);
setDifferenceMode(true);
setObservedObject(ObjectName);
setObservedAttribute("AttributeName");
```

To see all possible configurations of a `CounterMonitor` instance, refer to the JMX 1.0 API documentation, which you can download from http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html. The archive that you download includes the API documentation.
Configuring GaugeMonitor Objects

GaugeMonitor objects observe changes in MBean attributes that are expressed as integers or floating-point. The following list describes groups of GaugeMonitor operations that you use to achieve typical configurations of a GaugeMonitor instance:

- Sends a notification when the observed attribute is beyond the high threshold.
  ```java
  setHighThreshold(int Highthreshold);
  setNotifyHigh(true);
  setObservedObject(ObjectName);
  setObservedAttribute("AttributeName");
  ```

- Sends a notification when the observed attribute is outside the range of the high or low threshold.
  ```java
  setThresholds(int Highthreshold, Lowthreshold);
  setNotifyHigh(true);
  setNotifyLow(true);
  setObservedObject(ObjectName);
  setObservedAttribute("AttributeName");
  ```

- Sends a notification when the difference between two consecutive observations is outside the range of the high or low threshold.
  ```java
  setThresholds(int Highthreshold, Lowthreshold);
  setNotifyHigh(true);
  setNotifyLow(true);
  setDifferenceMode(true);
  setObservedObject(ObjectName);
  setObservedAttribute("AttributeName");
  ```

GaugeMonitor does not support an offset or modulus.

To see all possible configurations of a GaugeMonitor instance, refer to the JMX 1.0 API documentation, which you can download from [http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html](http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html). The archive that you download includes the API documentation.
Configuring StringMonitor Objects

StringMonitor objects observe changes in MBean attributes that are expressed as strings. The following list describes groups of StringMonitor operations that you use to achieve typical configurations of a StringMonitor instance:

- Sends a notification when the observed attribute matches the string specified in StringToCompare.
  
  ```java
  setStringToCompare(String);
  setNotifyMatch(true);
  setObservedObject(ObjectName);
  setObservedAttribute("AttributeName");
  ```

- Sends a notification when the observed attribute differs from the string specified in StringToCompare.
  
  ```java
  setStringToCompare(String);
  setNotifyDiffer(true);
  setObservedObject(ObjectName);
  setObservedAttribute("AttributeName");
  ```

To see all possible configurations of a StringMonitor instance, refer to the JMX 1.0 API documentation, which you can download from http://jcp.org/aboutJava/communityprocess/final/jsr003/index.html. The archive that you download includes the API documentation.
### Notification Types

Each type of monitor MBean emits specific types of notifications. The following table describes the type of notifications that monitor MBeans emit.

<table>
<thead>
<tr>
<th>A Monitor MBean of This Type</th>
<th>Emits This Notification Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CounterMonitor</td>
<td>A counter monitor emits a <code>jmx.monitor.counter.threshold</code> when the value of the counter reaches or exceeds a threshold known as the comparison level.</td>
</tr>
</tbody>
</table>

- **GaugeMonitor**
  - If the observed attribute value is **increasing** and becomes equal to or greater than the high threshold value, the monitor emits a notification type of `jmx.monitor.gauge.high`. Subsequent crossings of the high threshold value do not cause further notifications unless the attribute value becomes equal to or less than the low threshold value.
  - If the observed attribute value is **decreasing** and becomes equal to or less than the low threshold value, the monitor emits a notification type of `jmx.monitor.gauge.low`. Subsequent crossings of the low threshold value do not cause further notifications unless the attribute value becomes equal to or greater than the high threshold value.

| StringMonitor               | If the observed attribute value **matches** the string to compare value, the monitor emits a notification type of `jmx.monitor.string.matches`. Subsequent matches of the string to compare values do not cause further notifications unless the attribute value differs from the string to compare value. |

- If the attribute value **differs** from the string to compare value, the monitor emits a notification type of `jmx.monitor.string.differs`. Subsequent differences from the string to compare value do not cause further notifications unless the attribute value matches the string to compare value.
Error Notification Types

All monitors can emit the following notification types to indicate error cases:

- `jmx.monitor.error.mbean`, which indicates that the observed MBean is not registered in the MBean Server. The observed object name is provided in the notification.

- `jmx.monitor.error.attribute`, which indicates that the observed attribute does not exist in the observed object. The observed object name and observed attribute name are provided in the notification.

- `jmx.monitor.error.type`, which indicates that the object instance of the observed attribute value is `null` or not of the appropriate type for the given monitor. The observed object name and observed attribute name are provided in the notification.

- `jmx.monitor.error.runtime`, which contains exceptions that are thrown while trying to get the value of the observed attribute (for reasons other than the cases described above).

The counter and the gauge monitors can also emit the following `jmx.monitor.error.threshold` notification type under the following circumstances:

- For a counter monitor, when the threshold, the offset, or the modulus is not of the same type as the observed counter attribute.

- For a gauge monitor, when the low threshold or high threshold is not of the same type as the observed gauge attribute.

Sample Monitoring Scenarios

This section outlines some typical MBean attributes that you might consider monitoring to observe performance and/or resource usage. For more details on individual MBean attributes or methods, see the appropriate MBean API documentation.
### JDBC Monitoring

The JDBCConnectionPoolRuntime MBean maintains several attributes that describe the connections to a deployed JDBC connection pool. Applications may monitor these attributes to observe the connection delays and connection failures, as well as connection leaks. The following table outlines those MBean attributes typically used for JDBC monitoring.

<table>
<thead>
<tr>
<th>JDBCConnectionPoolRuntime MBean Attribute</th>
<th>Typical Monitoring Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeakedConnectionCount</td>
<td>Notify a listener when the total number of leaked connections reaches a predefined threshold. Leaked connections are connections that have been checked out but never returned to the connection pool via a close() call; it is important to monitor the total number of leaked connections, as a leaked connection cannot be used to fulfill later connection requests.</td>
</tr>
<tr>
<td>ActiveConnectionsCurrentCount</td>
<td>Notify a listener when the current number of active connections to a specified JDBC connection pool reaches a predefined threshold.</td>
</tr>
<tr>
<td>ConnectionDelayTime</td>
<td>Notify a listener when the average time to connect to a connection pool exceeds a predefined threshold.</td>
</tr>
<tr>
<td>FailuresToReconnect</td>
<td>Notify a listener when the connection pool fails to reconnect to its datastore. Applications may notify a listener when this attribute increments, or when the attribute reaches a threshold, depending on the level of acceptable downtime.</td>
</tr>
</tbody>
</table>
Monitoring WebLogic Server MBeans
Index

A
ADMIN_JNDI_NAME JNDI variable 2-7
Administration Console
defined 1-21
Local Configuration MBeans 1-7
administration domain. See domain 1-3
Administration MBeanHome interface
defined 1-18
determining active domain and servers
5-2
retrieving ClusterRuntimeMBean 5-16
retrieving from an external client 2-7
retrieving ServerRuntimeMBean 5-9,
5-11
retrieving through JNDI 2-7
retrieving with the Helper API 2-5
when to use 2-3
Administration MBeans
accessing from Administration Console
1-21
accessing from type-safe interface 3-6
accessing from weblogic.Admin 1-23
API documentation 1-11
defined 1-6
initializing Local Configuration MBeans
3-1
interfaces for accessing 2-3
lifecycle 1-8-1-11
Managed Server Independence 1-11
retrieving a list of 2-12-2-15
WebLogicObjectName 2-16
Administration Servers 1-5-1-11
accessing MBeans 1-18, 2-2
defined 1-3
JNDI tree 2-6
LogMBeans 2-20
registered MBeans 1-16
AttributeAddNotification object 4-2
AttributeChangeNotification object
4-2
AttributeRemoveNotification object
4-2

C
child relationship with MBeans 2-18
clusters 5-16
cfg.xml file 1-6-1-11
editing from Administration Console
1-21
no runtime data 1-12
configurable MBean attributes. See dynamic
changes to MBeans
Configuration MBeans
defined 1-3
See also Local Configuration MBeans
and Administration MBeans
CounterMonitor objects
configuring 6-9
instantiating for a remote application 6-7
type of data monitored 6-3
type of notifications emitted 6-13
ctx.lookup method 2-9
custom MBeans 1-16

D
derived gauge, defined 6-1
destroying MBeans 1-8
DifferenceMode attribute
  for CounterMonitor objects 6-10
  for GaugeMonitor objects 6-11
domains
  accessing all MBeans 1-18, 2-2
  defined 1-3
  determining current 5-2
  retrieving all MBeans 2-12
  saving configuration data in
    config.xml 1-6
  specified in WebLogicObjectName 2-16
dynamic attributes in the Administration
  Console 1-22
dynamic changes to MBeans 1-10

E
error notification types 6-14

G
GaugeMonitor objects
  configuring 6-11
  type of data monitored 6-4
  type of notifications emitted 6-13
getAllMBeans method 2-12
getMBeansByType method 2-15

H
handleNotification method 4-4
  for local applications 6-2
  for remote applications 4-4, 6-2
Helper API 2-4
  hierarchical relationship of MBeans 2-18

I
initial context 2-6
  instantiating MBeans 1-8
  Integer data type, monitoring 6-3

J
Javadoc
  for Configuration MBeans 1-11
  for Runtime MBeans 1-14
JDBC monitoring 6-15
JMS configuration data 3-6
JMX object names 2-16
JMX specification 1-2
JNDI tree
  Administration Servers 2-6
  Managed Servers 2-5

L
lifecycle of MBeans 1-8
lifecycle of servers, changing state 5-5
listen ports, setting 1-9
listeners
  creating 4-3, 6-2
  defined 4-1
  instantiating 6-5
  registering 4-6
  types of notification objects 6-13
Local Configuration MBeans
  accessing from MBeanServer interface 2-2
  accessing from weblogic.Admin 1-23
API documentation 1-11
changing with MBeanServer 3-4
changing with weblogic.Admin 3-3
defined 1-6
initialized from Administration MBeans 3-1
  interfaces for accessing 2-3
  lifecycle 1-8–1-11
no access from Administration Console 1-21
on Administration Server 1-16
retrieving a list of 2-12–2-15
WebLogicObjectName 2-16
WebLogicObjectName, examples 2-22
Local MBeanHome interface
defined 1-18
retrieving from an internal client 2-9
retrieving ServerRuntimeMBean 5-6
retrieving through JNDI 2-6
retrieving with the Helper API 2-4
when to use 2-3
LOCAL_JNDI_NAME JNDI variable 2-6
log messages 4-2
LogMBean on Administration Servers 2-20

M
managed resources, defined 1-2
Managed Server Independence (MSI) 1-11
Managed Servers
defined 1-3
JNDI tree 2-5
local interface, performance of 1-18, 2-3
MBean replicas 1-5, 1-8
MBeans accessible from 1-16, 1-18
propagating changes to Local Configuration MBeans 1-10
runtime information about clusters 5-16
See also Local MBeanHome interface
MBean types, defined 2-17
MBeanHome interface 1-17
See also Local MBeanHome interface,
Administration MBeanHome interface, and type-safe interface
MBeanHome methods. See type-safe interface
MBeans
accessing, main steps 2-2
creating custom 1-16
defined 1-2
notifications generated 4-2
See also Local Configuration MBeans,
Administration MBeans, and Runtime MBeans
MBeanServer interface
accessing MBeans 2-10
changing configuration data 3-4
defined 1-17
registering listeners 4-6
retrieving and changing runtime data 5-14
when to use 2-3
message level for standard out 3-3
metrics for runtime data 1-12
modulus for CounterMonitor objects 6-9
monitor MBeans
defined 6-1
instantiating 6-5
types 6-3
monitoring attributes of MBeans
comparing changes to MBean attributes 6-12
JDBC example 6-15
main steps 6-1
notification types 6-13
MSI 1-11

N
names of MBeans 2-16
notification listeners. See listeners
notifications
defined 4-1
types 6-13

O
object names for MBeans 2-13, 2-16
overriding values
in config.xml 1-10
of Administration MBeans 1-7

P
parent relationship with MBeans 2-18
performance metrics 1-12
persistence
of MBean modifications 1-6
of runtime data 1-12
propagating changes to Local Configuration MBeans 1-10

R
registering MBeans 1-16
remote applications 6-7
RemoteNotificationListener object 4-4, 6-2
replicas of Administration MBeans 1-8
RMI 1-18
runtime changes to MBeans 1-10, 1-22
Runtime MBeans
API documentation 1-14
defined 1-3
distribution 1-12
interfaces for accessing 2-3
on Administration Server 1-16
persistence 1-12
retrieving a list of 2-12–2-15
retrieving with Administration
MBeanHome.getMBeansByType 5-9
WebLogicObjectName 2-16
Runtime MBeans, accessing
from Administration Console 1-21
from Administration MBeanHome 2-13, 5-9
from Local MBeanHome 5-6
from MBeanServer 5-14
from weblogic.Admin 1-23

S
saving modifications to MBeans 1-6
security MBeans 1-15
server RuntimeMBean interface
accessing from Administration
MBeanHome 5-9
changing with MBeanServer 5-14
defined 5-5
servers
changing state to SHUTDOWN 5-9
determining active 5-2
standard out
configuring message level with
MBeanServer 3-4
configuring message level with
weblogic.Admin 3-3
String data type, monitoring 6-4
StringMonitor objects
configuring 6-12
type of data monitored 6-4
type of notifications emitted 6-13

T
thresholds
for CounterMonitor objects 6-9
for GaugeMonitor objects 6-11
type, MBean 2-17
type-safe interface
accessing MBeans 2-12–2-15
defined 1-17
retrieving configuration data 3-6
when to use 2-3

W
weblogic.Admin utility
changing configuration data 3-3
defined 1-23
determining active domain and servers 5-4
finding WebLogicObjectName 2-19
weblogic.Server startup command 1-7, 1-8
WebLogicObjectName
defined 2-16
examples 2-22
finding with weblogic.Admin 2-19
retrieving with
   WebLogicMBean.getName
       2-13
using to retrieve ServerRuntimeMBean
       5-11