Sun Java Enterprise System 5 Monitoring Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819–5081–10
March 2007

070217@16599

Contents

Preface

This book describes the new monitoring feature in Sun Java™ Enterprise System 5 (Java ES). Monitoring is implemented by the Sun Java System Monitoring Framework 2.0 and the Sun Java System Monitoring Console 1.0.

The procedures in this guide show you how to configure and enable the Monitoring Framework for each of your installed components, then how to view all monitored data in the Monitoring Console. This guide does not document log files nor other monitoring mechanisms of individual components outside this framework.

Who Should Use This Book

This book is intended for the following audiences:

- Software architects who need to design a maintenance plan for Java ES deployments.
- System administrators who perform Java ES installation and configuration.
- System administrators and technicians who monitor and maintain Java ES deployments.

Before You Read This Book

You should be familiar with the documents of the Java ES documentation set described in the next section. You should also be familiar with the design and functioning of the Java ES components you wish to monitor.

Furthermore, if you are intending to install and configure the monitoring components, you must first complete the installation of all other components. Before you perform any installation or configuration, you should consult the *Sun Java Enterprise System 5 Release Notes for UNIX*.

Java ES Documentation Set

The Java ES documentation set describes deployment planning and system installation. The URL for system documentation is `http://docs.sun.com/coll/1286.2`. For an introduction to Java ES, refer to the books in the order in which they are listed in the following table.

TABLE P–1 Java Enterprise System Documentation

Document Title	Contents
Sun Java Enterprise System 5 Release Notes for UNIX *Sun Java Enterprise System 5 Release Notes for Microsoft Windows*	Contains the latest information about Java ES, including known problems. In addition, components have their own release notes listed in the Release Notes Collection (`http://docs.sun.com/coll/1315.2`).
Sun Java Enterprise System 5 Technical Overview	Introduces the technical and conceptual foundations of Java ES. Describes components, the architecture, processes, and features.
Sun Java Enterprise System 2005Q4 Deployment Planning Guide	Provides an introduction to planning and designing enterprise deployment solutions based on Java ES. Presents basic concepts and principles of deployment planning and design, discusses the solution life cycle, and provides high-level examples and strategies to use when planning solutions based on Java ES.
Sun Java Enterprise System 5 Installation Planning Guide	Helps you develop the implementation specifications for the hardware, operating system, and network aspects of your Java ES deployment. Describes issues such as component dependencies to address in your installation and configuration plan.
Sun Java Enterprise System 5 Installation Guide for UNIX *Sun Java Enterprise System 5 Installation Guide for Microsoft Windows*	Guides you through the process of installing Java ES. Also shows how to configure components after installation, and verify that they function properly.
Sun Java Enterprise System 5 Installation Reference for UNIX	Gives additional information about configuration parameters, provides worksheets to use in your configuration planning, and lists reference material such as default directories and port numbers on the Solaris Operating System and Linux operating environment.
Sun Java Enterprise System 5 Upgrade Guide for UNIX *Sun Java Enterprise System 5 Upgrade Guide for Microsoft Windows*	Provides instructions for upgrading to Java ES 5 from previously installed versions.
Sun Java Enterprise System 5 Monitoring Guide	Gives instructions for setting up the Monitoring Framework for each product component and using the Monitoring Console to view real-time data and create monitoring rules.

TABLE P–1 Java Enterprise System Documentation *(Continued)*

Document Title	Contents
Sun Java Enterprise System Glossary	Defines terms that are used in Java ES documentation.

Default Paths and File Names

The following table describes the default paths and file names of the Java ES components that implement monitoring.

TABLE P–2 Default Paths and File Names

Placeholder	Description	Default Value
mfwk-base	Represents the directory where the Monitoring Framework shared component is automatically installed. This path is also used as part of the configuration directory.	Solaris systems: `/opt/SUNWmfwk` Linux systems: `/opt/sun/mfwk`
MConsole-base	Represents the installation directory chosen for the Monitoring Console.	Solaris systems: `/opt/SUNWjesmc` Linux systems: `/opt/sun/jesmc`
WebConsole-base	Represents the directory where the Web Console shared component is automatically installed.	Solaris systems: `/etc/webconsole/console` Linux systems: `/etc/opt/webconsole/console`
AccessMgr-base	Represents the installation directory chosen for Sun Java System Access Manager.	Solaris systems: `/opt/SUNWam` Linux systems: `/opt/sun/identity`
AppServer-base	Represents the installation directory chosen for Sun Java System Application Server.	Solaris systems: `/opt/SUNWappserver/appserver` Linux systems: `/opt/sun/appserver`
CalServ-base	Represents the installation directory chosen for Sun Java System Calendar Server.	Solaris systems: `/opt/SUNWics5` Linux systems: `/opt/sun/calendar`
DirServ-base	Represents the installation directory chosen forSun Java System Directory Server.	Solaris systems: `/opt/SUNWdsee/ds6` Linux systems: `/opt/sun/ds6`
IM-base	Represents the installation directory chosen for Sun Java System Instant Messaging.	Solaris systems: `/opt/SUNWiim` Linux systems: `/opt/sun/im`

TABLE P–2 Default Paths and File Names *(Continued)*

Placeholder	Description	Default Value
MsgServ-base	Represents the installation directory chosen for Sun Java System Messaging Server.	Solaris systems: `/opt/SUNWmsgsr` Linux systems: `/opt/sun/messaging`
Portal-base	Represents the installation directory chosen for Sun Java System Portal Server.	Solaris systems: `/opt/SUNWportal` Linux systems: `/opt/sun/portal`
WebServer-base	Represents the installation directory chosen for Sun Java SystemWeb Server.	Solaris systems: `/opt/SUNWwbsvr7` Linux systems: `/opt/sun/webserver`

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–3 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	A placeholder to be replaced with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file.

Shell Prompts in Command Examples

The following table shows default system prompts and superuser prompts.

TABLE P–4 Shell Prompts

Shell	Prompt
C shell on UNIX and Linux systems	`machine_name%`
C shell superuser on UNIX and Linux systems	`machine_name#`
Bourne shell and Korn shell on UNIX and Linux systems	`$`
Bourne shell and Korn shell superuser on UNIX and Linux systems	`#`
Microsoft Windows command line	`C:\`

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P–5 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

Searching Sun Product Documentation

Besides searching Sun product documentation from the docs.sun.com[SM] web site, you can use a search engine by typing the following syntax in the search field:

search-term `site:docs.sun.com`

For example, to search for "broker," type the following:

```
broker site:docs.sun.com
```

To include other Sun web sites in your search (for example, java.sun.com, www.sun.com, and developers.sun.com), use `sun.com` in place of `docs.sun.com` in the search field.

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to `http://docs.sun.com` and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is 819-5081.

◆ ◆ ◆ CHAPTER 1

Overview of Java ES Monitoring

This book describes the Monitoring Framework 2.0 and Monitoring Console 1.0 components of the Sun Java™ Enterprise System (Java ES). Together, these components implement the new monitoring feature introduced in release 5.

The procedures in this guide show you how to enable the Monitoring Framework for each of your installed components, then how to view all monitored data in the Monitoring Console. This guide does not document log files, error messages, nor other monitoring mechanisms that individual components may implement outside the framework. Neither the Monitoring Framework nor the Monitoring Console provide management or administration capabilities for monitored components. For information about the administration of a component, see that product's own documentation.

This chapter introduces monitoring concepts and presents the architecture of the Monitoring Framework.

This chapter contains the following sections:

The Monitoring Framework and the Monitoring Console Components

The Sun Java System Monitoring Framework provides the infrastructure to instrument components and expose their attributes for observation. It defines a hierarchy of monitored objects called the *Common Monitoring Model (CMM)*, based on the industry standard Common Information Model (CIM) specification. Each product component exposes the objects that represent its observable attributes, and the *node agent* aggregates a view of multiple components on one host. The Monitoring Framework also provides the mechanism to gather operational statistics and define alarms based on user-defined thresholds.

The Sun Java System Monitoring Console is the graphical interface for monitoring Java ES components. It includes a *master agent* that connects to all node agents in a Java ES deployment. The Monitoring Console is a web-based application that relies on the Sun Java SystemWeb Console to be accessible anywhere through HTTP. On the main screen, it provides a summary status of all enabled components, including any alarms that were triggered. You can then access the hierarchy of monitored objects in each component and see the detailed status and the real-time values of all monitored attributes. The Monitoring Console interface allows you to display the details of any alarm, acknowledge it, or create new monitoring rules based on any attribute.

How Java ES Monitoring Works

Monitoring is the entire process of gathering runtime data, exposing it, and computing quality of service criteria so that the system administrator can assess performance and be notified of alarms During runtime operation, administrators only need to interact with the Monitoring Console to view performance statistics, create rules to monitor automatically, and acknowledge alarms. However, for configuration, troubleshooting and advanced monitoring, it helps to understand the architecture of theMonitoring Framework and how it connects to the Monitoring Console.

Monitoring in Java ES is based on the following concepts:

- The Common Monitoring Model (CMM) ensures that all Java ES components expose uniform objects and values for comparable attributes.
- Java objects defined by the CMM interfaces provide standardized instrumentation for product components.
- A node agent exposes all monitored objects for all components installed on a system and manages the statistics, rules, and alarms for those objects.
- A master agent on a separate host aggregates all monitored objects from all node agents and makes the data available to the Monitoring Console.

The following sections explain each of these concepts of the monitoring architecture in more detail.

The Common Monitoring Model (CMM)

The basis of a standardized monitoring mechanism is the definition of what objects are monitored and the adoption of these objects across all monitored components. To this end, the monitoring architecture defines the Common Monitoring Model (CMM) as an extension of the Common Information Model (CIM) maintained by the Distributed Management Task Force (DMTF). CMM is both an information model specifying monitored objects such as computer, application, and so on, and a data model specifying uniform values such as the operational

status values. As part of the information model, CMM also defines the attributes of an object, for example the number of requests handled by a service, and relations between objects, such as the fact that a service is hosted on a certain computer.

Thanks to CMM, concepts such as applications, services, points of access, and so on are the same for all product components, even if the underlying implementation is different. For example, Web Server might expose a service that handles HTTP requests while Directory Server might expose a service that handles LDAP requests. However the standard object will capture what is common to these two functions, for example the ability to measure the number of requests handled, the average time to respond to a request over a given time period, and so on.

Furthermore, certain data values are standardized so that their meaning is uniform across the entire system. For example, the operational status `DEGRADED` always means that a service is still available but performance has dropped significantly, no matter which product component is being monitored.

The CMM specification is embodied in the Java interfaces and classes used for the instrumentation, which are described in Appendix A.

CMM Instrumentation

In the Monitoring Framework, the instrumentation is a set of Java interfaces and classes that implement the CMM definitions. For the new monitoring functionality in Java ES, the product components have instrumented their code to instantiate the CMM objects and expose run-time values through the attributes of the monitored objects. The CMM objects that are implemented by each component determines what can be monitored, and for this reason, some components expose less attributes than others. The list of objects and attributes that are exposed for monitoring by each product component is given in Appendix B.

Node Agents

In monitoring terminology, a node is a single logical host identified by a unique fully qualified domain name or IP address. A node can be either an entire system or a Solaris zone configured as a virtual system. The node agent communicates with all instrumented components on that host and exposes all of their monitored objects. The node agent also manages all of the logic to collect performance statistics, monitor thresholds defined in rules, and generate alarms for the monitored objects it contains.

The following diagram represents the contents of a node agent on a single host that has instances of three Java ES product components. It also shows how the instrumentation is instantiated in the node agent to expose values provided by the product components.

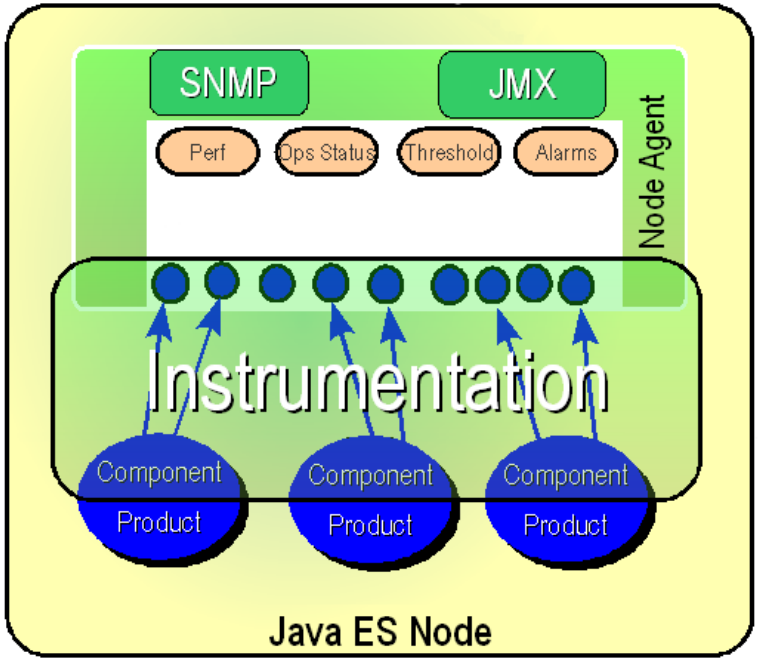


FIGURE 1–1 Diagram of a Node Agent

The node agent is implemented as a module loaded into the Common Agent Container , which is itself a Java Virtual Machine. The implementation of the node agent is based on the Java Management Extensions (JMX), the standard Java extension for monitoring and remote management. Any JMX-enabled monitoring application that understands CMM can access the monitored objects in the node agent. Using JMX functionality, the node agent can also expose certain monitored objects through the Simple Network Monitoring Protocol (SNMP).

The Master Agent

The master agent is deployed on a separate machine as part of the Monitoring Console installation. The master agent is configured with the name or address of all nodes so that it can aggregate the monitored objects from all of the node agents. The master agent is also based on JMX, which it uses to communicate with the node agents, and is also loaded into its local Common Agent Container.

The following diagram represents a master agent connected to two nodes. The Monitoring Console connects to the master agent to monitor the three components on each node. If you wish to use SNMP for monitoring, you must connect to each node separately, because the master agent does not aggregate SNMP attributes. The master agent is designed for use with the Monitoring Console only and cannot be accessed by other monitoring applications.

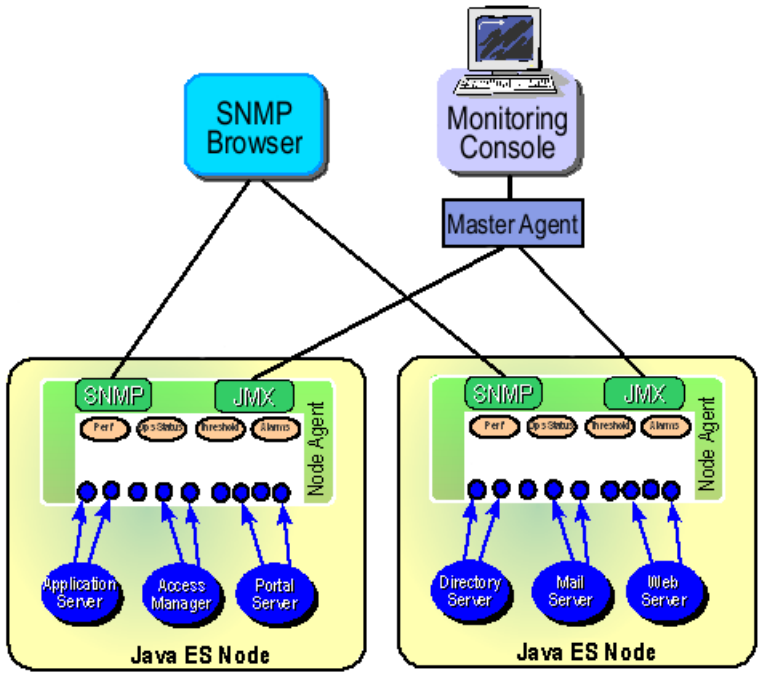


FIGURE 1–2 Diagram of the Overall Monitoring Architecture

Suggested Installation Sequence

If you choose to evaluate or deploy the monitoring feature of Java ES, it is easiest to perform the installation in the following order:

1. Install and configure all of the components in your deployment according to the recommendations and instructions in the *Sun Java Enterprise System 5 Installation Guide for UNIX*.
2. Enable and configure the Monitoring Framework for all of your monitored components, as described in Chapter 2.
3. Install the Monitoring Console on a separate host, start the master agent, and then start the web server, as described in Chapter 3. All of your monitored components should then be visible and actively monitored in the Monitoring Console.

Note – Due to an incompatibility of the node agent and master agent in this release, the Monitoring Console must be installed on a host that does not contain any other Java ES components. See "Troubleshooting the Monitoring Console" on page 65 for more details.

Whenever you modify your deployed components after enabling monitoring, you will need to restart the container for the master agent and the web server for the Monitoring Console as described in "Troubleshooting the Monitoring Framework" on page 27.

◆ ◆ ◆ CHAPTER 2

Enabling and Configuring the Monitoring Framework

As described in "How Java ES Monitoring Works" on page 14, the Monitoring Framework provides the instrumentation and the node agent needed by every monitored component. Therefore, the Monitoring Framework is a shared component that is automatically installed whenever you install a monitored component using the Java Enterprise System installer.

However, many of the monitored components do not have monitoring enabled by default, and some require further configuration to make them appear in the node agent. Follow the procedures in this chapter for each of the product components that you have installed.

Note – It is good practice to install and configure all product components that you intend to run on a given host before performing any of the procedures in this chapter. Before you perform any installation or configuration, you should consult the *Sun Java Enterprise System 5 Release Notes for UNIX*.

These procedures use the `mfwksetup` command that is usually not needed and which therefore remains undocumented.

This chapter contains the following sections:

Installed Directory Layout

As a shared component, the Monitoring Framework is automatically installed whenever it is needed. For the name of the package installed on your operating system, see Chapter 5, "List of Installable Packages," in *Sun Java Enterprise System 5 Installation Reference for UNIX*. The following table describes the directories in the Monitoring Framework package. The default installation directory *mfwk-base* has the following meaning, as described in "Default Paths and File Names" on page 9:

- Solaris systems: `/opt/SUNWmfwk`
- Linux systems: `/opt/sun/mfwk`

TABLE 2–1 Directories Used by the Monitoring Framework

Path	Description of contents
mfwk-base`/config`	Template for a configuration file
Solaris systems: *mfwk-base*`/lib`	Java archive (`.jar`) files
Linux systems: *mfwk-base*`/share/lib`	
Solaris systems: *mfwk-base*`/lib`	32–bit runtime library files (`.so`)
Linux systems: *mfwk-base*`/share/lib`	
Solaris SPARC® systems: *mfwk-base*`/lib/sparcv9`	64–bit runtime library files (`.so`)
Solaris x86 systems: *mfwk-base*`/amd64`	
Linux systems: *mfwk-base*`/lib64`	
mfwk-base`/bin`	Public scripts and private binaries
mfwk-base`/mib`	Text versions of SNMP MIBs supported by the Monitoring Framework
mfwk-base`/xml`	Common Agent Container descriptor templates for agent and master agent (deployed by the `mfwksetup` command)
mfwk-base`/dtd`	DTD files for the OSS/J functionality.
`/etc/`*mfwk-base*`/config`	Configuration files, including security-related ones
`/etc/`*mfwk-base*`/xml`	Common Agent Container descriptors for agents and examples
`/var/`*mfwk-base*`/logs`	Log files of the Monitoring Framework
`/var/`*mfwk-base*`/reports`	Base directory for monitoring rule reports
`/var/`*mfwk-base*`/alarms`	Repository for alarm files

Using the Monitoring Framework with Access Manager

By default, monitoring is enabled in Access Manager, but a limitation prevents the monitored objects from appearing in the Monitoring Console.

See "Instrumentation of Access Manager" on page 69 for the list of objects and attributes you can monitor.

▼ To Enable Monitoring in Access Manager

1. **Temporarily disable monitoring in Access Manager with the following commands:**

   ```
   cacaoadm unregister-module com.sun.cmm.am.xml
   cacaoadm restart
   ```

2. **Open the Access Manager XML descriptor file for editing:**

   ```
   vi /etc/AccessMgr-base/config/com.sun.cmm.am.xml
   ```

3. **Find the lines containing:**

   ```
   <param-name>Product Name</param-name>
   <param-value>Access Manager</param-value>
   ```

   and modify the second line to:

   ```
   <param-value>Java ES Access Manager</param-value>
   ```

   Save the file and exit the editor.

4. **Register the modified XML module:**

   ```
   mfwk-base/bin/mfwksetup -u /etc/AccessMgr-base/config/com.sun.cmm.am.xml
   mfwk-base/bin/mfwksetup -r /etc/AccessMgr-base/config/com.sun.cmm.am.xml
   ```

5. **Restart the Common Agent Container:**

   ```
   cacaoadm restart
   ```

Troubleshooting Due to untested behavior with third-party web containers, monitoring is disabled by default when Access Manager is deployed in Websphere or Weblogic. You may enable monitoring as described in how "To Selectively Disable and Re-Enable Monitoring" on page 56, although this configuration is unsupported.

Using the Monitoring Framework with Application Server

See "Instrumentation of Application Server" on page 69 for the list of objects and attributes you can monitor.

▼ To Enable Monitoring in Application Server

1 **Edit the file** `/var/`*AppServer-base*`/domains/domain1/config/domain.xml` **and change all** `module-monitoring-level` **settings from** `OFF` **to** `HIGH`**. Alternatively:**

   a. **Log onto the Application Server administration console at** `https://`*hostname*`:4849`

   b. **Select Configurations, then select server-config (Admin Config)**

   c. **Set the Monitoring value to** `HIGH`

   d. **Set all other values to** `HIGH`

2 **Restart Application Server with the following commands:**

```
cd AppServer-base/appserv/bin
asadmin stop-domain domain1
asadmin start-domain user myUser domain1
```

   Enter the password for *myUser* when prompted.

3 **If you have deployed and monitored an instance of Portal Server with Application Server, the process of restarting Application Server interferes with Portal Server monitoring. To make the Portal Server instance appear in the Monitoring Console, you must visit a portal page in a browser. For example, load the page** `http://portalserv.example.com:8080/portal` **to allow monitoring of** `portalserv.example.com`**.**

Troubleshooting Due to a limitation, the monitored objects for Application Server are removed from the Monitoring Framework when Application Server crashes or is down. When this happens, Application Server disappears from the Monitoring Console and can no longer be monitored.

Using the Monitoring Framework with Calendar Server

See "Instrumentation of Calendar Server" on page 69 for the list of objects and attributes you can monitor.

▼ To Enable Monitoring in Calendar Server

1 **Edit the `ics.conf` file:**

```
vi CalServ-base/cal/config/ics.conf
```

2 **Add the line:**

```
local.mfagent.enable="yes"
```

3 **Register Calendar Server XML module:**

```
mfwk-base/bin/mfwksetup -r /opt/SUNWics5/cal/lib/com.sun.cmm.cs.xml
```

4 **Set the `LD_LIBRARY_PATH` environment variable as follows:**

```
LD_LIBRARY_PATH=mfwk-base/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

5 **Restart Calendar Server:**

```
cd CalServ-base/cal/sbin/
./stop-cal
./start-cal
```

6 **Restart the Common Agent Container:**

```
cacaoadm restart
```

Using the Monitoring Framework with Directory Server

See "Instrumentation of Directory Server" on page 70 for the list of objects and attributes you can monitor.

▼ To Enable Monitoring in Directory Server

1 **Create a temporary password file:**

```
echo –n password > /tmp/pwd
```

2 **Enable the Monitoring Plugin with the following command:**

```
DirServ-base/ds6/bin/dscfg enable-plugin -e -p 389 -w /tmp/pwd  "Monitoring Plugin"
```

3 **Restart Directory Server:**

```
cd DirServ-base/ds6/bin
./dsadm restart /var/DirServ-base/DSinstance/
```

Using the Monitoring Framework with Instant Messaging

See "Instrumentation of Instant Messaging" on page 70 for the list of objects and attributes you can monitor.

▼ To Enable Monitoring with Instant Messaging

1 **Open the Instant Messaging XML descriptor file for editing:**

```
vi /etc/IM-base/default/com.sun.cmm.im.xml
```

2 **Change the Install Location from *IM-base* to `/etc/IM-base/default`.**

3 **Register the modified Instant Messaging XML descriptor:**

```
mfwk-base/bin/mfwksetup -r /etc/IM-base/default/com.sun.cmm.im.xml
```

4 **Enable Instrumentation by adding the following line to the file *IM-base*`/config/iim.conf`:**

```
iim_server.monitor.enable = true
```

5 **Restart Instant Messaging with the following commands:**

```
cd IM-base/sbin
./imadmin stop
./imadmin start
```

6 **Restart the Common Agent Container:**

```
cacaoadm restart
```

Using the Monitoring Framework with Messaging Server

See "Instrumentation of Messaging Server" on page 70 for the list of objects and attributes you can monitor.

▼ To Enable Monitoring in Messaging Server

1 **Enable Instrumentation with the following command:**

```
MsgServ-base/sbin/configutil -o local.mfagent.enable -v 1
```

2. **Register the Messaging Server XML module:**

   mfwk-base`/bin/mfwksetup -r` *MsgServ-base*`/lib/com.sun.cmm.ms.xml`

3. **Restart Messaging Server:**

   ```
   cd MsgServ-base/sbin
   ./stop-msg
   ./start-msg
   ```

4. **Restart the Common Agent Container:**

   ```
   cacaoadm restart
   ```

Using the Monitoring Framework with Portal Server

See "Instrumentation of Portal Server" on page 70 for the list of objects and attributes you can monitor.

▼ To Enable Monitoring in Portal Server

◗ **To enable Portal Server, the user has to log onto**

`http://`*FullHostname*`:8080/portal/dt`

This will compile the portal JSP, which creates the portal instance that is ready for monitoring.

Troubleshooting Whenever the Application Server hosting the Portal Server is restarted, you must manually re-enable monitoring with this procedure.

Using the Monitoring Framework with Web Server

See "Instrumentation of Web Server" on page 70 for the list of objects and attributes you can monitor.

▼ To Enable Monitoring in Web Server

1. **Start Web Server with the following command:**

   ```
   cd /var/WebServer-base/https-FullHostname/bin
   ./startserv
   ```

2. **Start the administration server:**

   ```
   cd /var/WebServer-base/admin-server/bin
   ./startserv
   ```

Setting up the Common Agent Container

The Common Agent Container is another shared component and one that the Monitoring Framework depends on to run the node agent. Depending on your installation sequence, Common Agent Container may be stopped and need restarting. In addition, Common Agent Container has been instrumented and can be monitored as well. For a description of the monitored objects, see "Instrumentation of the Common Agent Container" on page 69.

To check if the Common Agent Container and thus the node agent is already started, run the following command:

```
cacaoadm status
```

If a message similar to the following appears, the node agent is running:

```
default instance is DISABLED at system startup.
Smf monitoring process:
26996
Uptime: 0 day(s), 0:57
```

If a message similar to the following appears, the node agent is not running:

```
default instance is DISABLED at system startup.
default instance is not running.
```

▼ To Enable Monitoring of the Common Agent Container

The Common Agent Container is a shared component that has instrumentation to allow monitoring. As described in "Node Agents" on page 15, all Java ES components on a host or in a zone share the Common Agent Container and the node agent. Perform this task as root on every logical host in your deployment where you wish to monitor the Common Agent Container.

1. **If the Common Agent Container is running, stop it with the following command:**

   ```
   cacaoadm stop
   ```

2. **Enable instrumentation of the container itself:**

   ```
   cacaoadm set-param enable-instrumentation=true
   ```

3. **Check the value of the parameter you just set and restart the Common Agent Container:**

   ```
   cacaoadm get-param enable-instrumentation
   cacaoadm start
   ```

4. **Create a key password:**

   ```
   echo –n password > /etc/mfwk-base/config/security/password.cacao
   ```

5 **Generate your key:**

```
mfwk-base/bin/cpgenkey -n cacao -p /etc/mfwk-base/config/security/password.cacao
```

6 **Register the Common Agent Container's own monitoring modules:**

```
cacaoadm register-module /usr/lib/cacao/ext/instrum/config/com.sun.cacao.instrum.xml
cacaoadm register-module /usr/lib/cacao/ext/instrum_jesmf/config/com.sun.cacao.instrum.jesmf.xml
cacaoadm register-module /usr/lib/cacao/ext/instrum_jesmf/config/com.sun.cacao.cmm.xml
```

Troubleshooting the Monitoring Framework

See also the known issues listed in the *Sun Java Enterprise System 5 Release Notes for UNIX*.

Using the Monitoring Framework on HP-UX Platforms

The Java Virtual Machine (JVM) on HP-UX is not tuned by default for the task-intensive processing needed by Monitoring Framework, and this may lead to `OutOfMemory` exceptions. To configure the JVM, download and run the `HPjconfig` tool from the following location:`http://h21007.ww2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/1,1701,1620,00.html`.

Using the Monitoring Framework on Microsoft Windows

Monitoring Java ES components on the Windows platform through the Monitoring Framework is fully supported, although there are some differences. For example, you must upgrade to Java 1.5 or later to avoid certain known issues. For other known issues, see the *Sun Java Enterprise System 5 Release Notes for Microsoft Windows*.

▼ To Restart a Node Agent

If you need to restart the Common Agent Container that hosts a node agent, the components monitored through that node agent will not be visible in the Monitoring Console until you perform this procedure:

1 **Restart the Common Agent Container that hosts the *node* agent:**

```
cacaoadm restart
```

2 **Restart the Common Agent Container that hosts the *master* agent. The master agent runs in the Monitoring Framework on the host or in the zone where you installed theMonitoring Console.**

```
cacaoadm restart
```

The master agent will automatically reconnect with all node agents that it was previously monitoring.

3 **Restart the web server that hosts the Monitoring Console:**

```
/usr/sbin/smcwebserver restart
```

The mfwkadm Command

This section reproduces the man page for the mfwkadm command, a maintenance command in man page section 1M. Use this command to manage the contents of a node agent, including all of the modules for components being monitored and any monitoring rules, also known as jobs, that you have defined on this node. Some of the terms and descriptions in the man page have been modified here to match those used in this document.

Synopsis

mfwkadm --help

mfwkadm start

mfwkadm stop

mfwkadm restart

mfwkadm list-params

mfwkadm list-modules

mfwkadm info *runningInstance*

Performance Monitoring

mfwkadm pm-job observable-classes

mfwkadm pm-job observable-objects [class=*objectClass*] [domain=*objectDomain*]

mfwkadm pm-job observable-attributes class=*objectClass*

mfwkadm pm-job list

mfwkadm pm-job info *jobName*

mfwkadm pm-job create *jobName* granularity=*integerValue* object=*objectName* [object=*objectName* ...]

mfwkadm pm-job delete *jobName*

mfwkadm pm-job suspend *jobName*

mfwkadm pm-job resume *jobName*

Operational Status Monitoring

mfwkadm opstat-job observable-classes

mfwkadm opstat-job observable-objects [class=*objectClass*] [domain=*objectDomain*]

mfwkadm opstat-job observable-attributes class=*objectClass*

mfwkadm opstat-job list

mfwkadm opstat-job info *jobName*

mfwkadm opstat-job create *jobName* granularity=*integerValue* object=*objectName* [object=*objectName* ...]

mfwkadm opstat-job delete *jobName*

mfwkadm opstat-job suspend *jobName*

mfwkadm opstat-job resume *jobName*

Threshold Value Monitoring

mfwkadm thrsh-job observable-classes

mfwkadm thrsh-job observable-objects [class=*objectClass*] [domain=*objectDomain*]

mfwkadm thrsh-job observable-attributes class=*objectClass*

mfwkadm thrsh-job list

mfwkadm thrsh-job info *jobName*

mfwkadm thrsh-job create *jobName* granularity=*integerValue* attributeName=*attributeName* attributeType=*attributeType* thresholdValue=*thresholdValue* thresholdOffset=*offsetValue* thresholdDirection=[RISING | FALLING] object=*objectName*

mfwkadm thrsh-job delete *jobName*

mfwkadm thrsh-job suspend *jobName*

mfwkadm thrsh-job resume *jobName*

Description

The mfwkadm utility is the command line interface for managing the Monitoring Framework agent, also called the node agent. The node agent runs inside the Common Agent Container. The mfwkadm utility can be used to stop and restart the node agent and to manage the

monitoring jobs it performs. This command should be run from the same host where the node agent is running. The order of the arguments of this command presented here must be respected.

To change the language of the output messages, set the LC_MESSAGE environment variable to your *locale*. The mfwkadm command will use the messages contained in the file named JesmfMessages_*locale*.pm in the lib/resources directory. If the locale has no corresponding file of messages, or if no locale is specified, the mfwkadm command will use the default set of messages in the file JesmfMessages.pm.

The mfwkadm utility has the following subcommands. Those marked with an asterisk (*) require the Common Agent Container to be running and the node agent to be loaded:

- start
- stop
- restart
- list-params (*)
- list-modules (*)
- info (*)
- pm-job (*)
- opstat-job (*)
- thrsh-job (*)

Depending on the number of Common Agent Container modules to load, there is a delay of a few seconds to a few minutes between starting the node agent and the availability of the mfwkadm utility. During this period of time, commands fail with an explicit message.

Options

The following options are supported.

--help Display the usage summary.

Subcommands

start

Start the Monitoring Framework node agent and its associated component product modules without stopping the Common Agent Container.

This action first deploys the node agent and then deploys the associated component product modules in the Common Agent Container. This facility is a wrapper on top of the cacaoadm utility's lock and undeploy subcommands.

The start subcommand only starts the node agent and the Java ES component modules associated with the Monitoring Framework. Component modules have the prefix com.sun.cmm.

Security: The start subcommand can be run only by the user who launched the Common Agent Container. Otherwise an error message similar to the following will be displayed:

```
Error occured in mfwkadm
Problem running /usr/sbin/cacaoadm unlock com.sun.mfwk 2>&1.
Stdout/Stderr: This command must be run by user: [root].
```

stop

Stop the Monitoring Framework node agent and its associated Java ES component modules in the Common Agent Container.

This action first stops any Java ES component's modules deployed in the Common Agent Container, and then stops the node agent. This facility is a wrapper on top of the cacaoadm lock and unlock subcommands.

The stop subcommand stops only those Java ES component modules associated with the Monitoring Framework and then the node agent itself. Component modules have the prefix com.sun.cmm.

Security: The stop subcommand can be run only by the user who launched the Common Agent Container. Otherwise an error message similar to the following will be displayed:

```
Error occured in mfwkadm
Problem running /usr/sbin/cacaoadm unlock com.sun.mfwk 2>&1.
Stdout/Stderr: This command must be run by user: [root].
```

restart

Restart the Monitoring Framework node agent and its associated Java ES component modules in the Common Agent Container.

This action will attempt to stop and then start the node agent and its associated modules in the Common Agent Container in the same way as the stop and start subcommands.

Security: The restart subcommand can be run only by the user who launched the Common Agent Container. Otherwise an error message similar to the following will be displayed:

```
Error occured in mfwkadm
Problem running //usr/sbin/cacaoadm unlock com.sun.mfwk 2>&1.
Stdout/Stderr: This command must be run by user: [root].
```

list-params

List all the configuration parameters related to the Monitoring Framework node agent.

Security: There is no user restriction for this command.

`list-modules`
Display a list of those component product modules that implement the Common Monitoring Model (CMM) and that are loaded into the Common Agent Container. This subcommand also lists all running instances of each installed Java ES component. Each component can have zero, one, or more running instances.

Security: For users other than the one who launched the Common Agent Container, the list of installed Java ES components does not include component instances.

`info` *runningInstance*
Display information on the named *runningInstance*. The *runningInstance* must match a running instance listed in the output of the `list-modules` subcommand.

The displayed information includes:

- For each type of monitoring job, all observable objects associated with the running instance, sorted by their class name. Observable objects are those on which you can create a performance monitoring job, an operational status job, or a threshold monitoring job using the `pm-job`, `opstat-job`, or `thrsh-job` subcommands, respectively.
- For each class of observable objects, all of its observable attributes, including the name and type of each.

Security: For users other than the one who launched the Common Agent Container, no information will be displayed.

Performance Monitoring

`pm-job observable-classes`
Display the list of all currently observable classes of objects for which you can create performance monitoring jobs.

`pm-job observable-objects` [class=*objectC lass*] [domain=*objectDomain*]
Display the list of all currently observable objects for which you can create performance monitoring jobs. By default all objects of all observable classes and in every domain will be listed. The list of objects is sorted by their class name.

class=*objectClass*
Specifying the optional *objectClass* will restrict the output to observable objects of that specific class. The *objectClass* must be one of the classes listed by the `pm-job observable-classes` subcommand.

domain=*objectDomain*
Specifying the optional *objectDomain* will restrict the output to observable objects in that domain. The domain of an object is the string preceding the colon (":") character in an object's name.

`pm-job observable-attributes class=`*objectClass*
: Display the list of all observable attributes in the specified *objectClass*. Attributes are displayed with their name and type. The *objectClass* must be one of the classes that supports performance monitoring jobs, as listed by the `pm-job observable-classes` subcommand.

`pm-job list`
: Displays the list of all currently defined performance monitoring jobs. Jobs are listed for each object having a defined performance job, and objects are sorted by their class name. The information displayed for each job is the same as displayed by the `pm-job info` subcommand.

  Security: For other users than the one who launched the Common Agent Container, no jobs will be displayed.

`pm-job info` *jobName*

Display verbose information about a performance monitoring job named *jobName*. The *jobName* must be one displayed by the `pm-job list` subcommand. This subcommand displays the following information:

- The name of the performance monitoring job.
- The type of the performance monitoring job, either "by object" or "by class." Jobs by object monitor one or more named object instances, whereas jobs by class monitor every instance of an object class. Note that it is not possible to create jobs by class with the `mfwkadm` utility.
- The state of the performance monitoring job: active on-duty, active off-duty, or suspended. An active on-duty job is currently scheduled to run and is collecting data. An active off-duty job is running but not collecting data because the current time is out of its working schedule. A suspended job is not running nor collecting any data. Use the `pm-job suspend` and `pm-job resume` subcommands to change the running state of a performance monitoring job.
- The granularity in seconds of the performance monitoring job. This is the interval for data collection by this job.
- The reporting period of the monitoring job. The reporting period times the granularity equals the notification frequency. For example, if the granularity period is 10 seconds and the reporting period is 6, a job reporting by event will collect data every 10 seconds and will send a notification including 6 reports every 60 seconds (10*6). If the job is also reporting by file, it will send an event every 60 seconds containing the locations of the 6 generated files.
- Whether the performance monitoring job is reporting by event. This means the results of the performance monitoring job are sent as notifications to a registered client.
- Whether the performance monitoring job is reporting by file. This means the reports of the performance monitoring job are written in local files and notifications containing the filenames are sent to registered clients.

- The report format of the performance monitoring job, which is always XML.
- The schedule of the performance monitoring job. The schedule specifies what days and times the job is active on-duty or active off-duty (collecting data or not, respectively).

Then for a job by-object:

- The list of observed objects, ordered by name.
- If only a subset of observable attributes are specified, the observed attributes of the observed objects are listed by name and type.

And for a job by-class:

- The list of observed classes, ordered by name.
- If only a subset of observable attributes are specified, the observed attributes of the observed classes are listed by name and type. These attributes are common to all classes.

Security: For users other than the one who launched the Common Agent Container, no information will be displayed.

`pm-job create` *jobName* granularity=*integerValue* object=*objectName* [object=*objectName* ...]
Creates a new performance monitoring job on one or more objects. The `mfwkadm` command cannot create jobs by class. When creating performance monitoring jobs, the following parameters can be set:

jobName
A string uniquely identifying the performance monitoring job. The *jobName* cannot be already in use by any other performance monitoring job.

granularity=*integerValue*
The time specified in seconds between the initiation of two successive collections of measurement data, while the job is active on-duty. Examples of granularity period can be 300 seconds (5 minutes), 900 seconds (15 minutes), 1800 seconds (every half-hour), 3600 seconds (every hour). A granularity period of 300 seconds is sufficient in most cases. For some measurements it can be more meaningful to collect data with larger granularity periods.

object=*objectName* [object=*objectName* ...]
One or more observable objects that the performance monitoring job will collect data from and report on. The *objectName* must be one displayed by the `pm-job list` or `pm-job observable-objects` subcommands. Specifying multiple object=*objectName* parameters will create a single performance monitoring job that monitors multiple objects.

Security: This subcommand can be run only by the user who launched the Common Agent Container.

`pm-job delete` *jobName*
Delete a performance monitoring job named *jobName*. The *jobName* must be one displayed by the `pm-job list` subcommand.

Security: This subcommand can be run only by the user who launched the Common Agent Container.

`pm-job suspend` *jobName*
Suspend a performance monitoring job named *jobName*. A suspended job is not active and will no longer collect data, regardless of its schedule. However, the job remains defined and can be made active again with the `pm-job resume` subcommand. The *jobName* must be one displayed by the `pm-job list` subcommand.

Security: This subcommand can be run only by the user who launched the Common Agent Container.

`pm-job resume` *jobName*
Resume a performance monitoring job named *jobName*. A resumed job will begin collecting data and sending reports according to its schedule. The *jobName* must be one displayed by the `pm-job list` subcommand. This is the counterpart to the `pm-job suspend` subcommand.

Security: This subcommand can be run only by the user who launched the Common Agent Container.

Operational Status Monitoring

`opstat-job observable-classes`
Display the list of all currently observable classes of objects for which you can create operational status monitoring jobs.

`opstat-job observable-objects` [class=*objectClass*] [domain=*objectDomain*]
Display the list of all currently observable objects for which you can create operational status monitoring jobs. By default all objects of all observable classes and in every domain will be listed. The list of objects is sorted by their class name.

class=*objectClass*
Specifying the optional *objectClass* will restrict the output to observable objects of that specific class. The *objectClass* must be one of the classes listed by the `opstat-job observable-classes` subcommand.

domain=*objectDomain*
Specifying the optional *objectDomain* will restrict the output to observable objects in that domain. The domain of an object is the string preceding the colon (":") character in the object's name.

`opstat-job observable-attributes class=`*objectClass*
Display the list of all observable attributes in the specified *objectClass*. Attributes are displayed with their name and type. The *objectClass* must be one of the classes listed by the `opstat-job observable-classes` subcommand.

`opstat-job list`

Displays the list of all currently defined operational status monitoring jobs. Jobs are listed for each object having a defined operational status job, and objects are sorted by their class name. The information displayed for each job is the same as displayed by the `opstat-job info` subcommand.

Security: For other users than the one who launched the Common Agent Container, no jobs will be displayed.

`opstat-job info` *jobName*

Display verbose information about an operational status monitoring job named *jobName*. The *jobName* must be one displayed by the `opstat-job list` subcommand. This subcommand displays the following information:

- The name of the operational status monitoring job.
- The type of the operational status monitoring job, either "by object" or "by class." Jobs by object monitor a named object instance, whereas jobs by class monitor every instance of an object class. Note that it is not possible to create jobs by class with the `mfwkadm` utility.
- The state of the operational status monitoring job: active on-duty, active off-duty, or suspended. An active on-duty job is currently scheduled to run and is collecting data. An active off-duty job is running but not collecting data because the current time is out of its working schedule. A suspended job is not running nor collecting any data. Use the `opstat-job suspend` and `opstat-job resume` subcommands to change the running state of an operational status monitoring job.
- The granularity in seconds of the operational status monitoring job. This is the interval for data collection by this job.
- Whether the operational status monitoring job is reporting by event. This means the results of the operational status monitoring job are sent as notifications to a registered client.
- Whether the operational status monitoring job is reporting by file. This means the reports of the operational status monitoring job are written in local files and notifications containing the filenames are sent to registered clients.
- The report format of the operational status monitoring job, which is always `XML`.
- The schedule of the operational status monitoring job. The schedule specifies what days and times the job is active on-duty or active off-duty (collecting data or not, respectively).
- For a job by-object, the list of observed objects, ordered by name.
- For a job by-class, the list of observed classes, ordered by name.

Security: For users other than the one who launched the Common Agent Container, no information will be displayed.

`opstat-job create` *jobName* granularity=*integerValue* object=*objectName* [object=*objectName* ...]

Creates a new operational status monitoring job on one or more objects. The `mfwkadm` command cannot create jobs by class. When creating performance monitoring jobs, the following parameters can be set:

jobName

A string uniquely identifying the operational status monitoring job. The *jobName* cannot be already in use by any other operational status monitoring job.

granularity=*integerValue*

The time specified in seconds between the initiation of two successive collections of measurement data, while the job is active on-duty.

object=*objectName* [object=*objectName* ...]

One or more observable objects that the operational status monitoring job will collect data from and report on. The *objectName* must be one displayed by the `opstat-job list` or `opstat-job observable-objects` subcommands. Specifying multiple object=*objectName* parameters will create a single operational status job that monitors multiple objects.

Security: This subcommand can be run only by the user who launched the Common Agent Container.

`opstat-job delete` *jobName*

Delete an operational status monitoring job named *jobName*. The *jobName* must be one displayed by the `opstat-job list` subcommand.

Security: This subcommand can be run only by the user who launched the Common Agent Container.

`opstat-job suspend` *jobName*

Suspend an operational status monitoring job named *jobName*. A suspended job is not active and will no longer collect data, regardless of its schedule. However, the job remains defined and can be made active again with the `opstat-job resume` subcommand. The *jobName* must be one displayed by the `opstat-job list` subcommand.

Security: This subcommand can be run only by the user who launched the Common Agent Container.

`opstat-job resume` *jobName*

Resume an operational status monitoring job named *jobName*. A resumed job will begin collecting data and sending reports according to its schedule. The *jobName* must be one displayed by the `opstat-job list` subcommand. This is the counterpart to the `opstat-job suspend` subcommand.

Security: This subcommand can be run only by the user who launched the Common Agent Container.

Threshold Value Monitoring

`thrsh-job observable-classes`
Display the list of all currently observable classes of objects for which you can create threshold monitoring jobs.

`thrsh-job observable-objects` [class=*objectClass*] [domain=*objectDomain*]
Display the list of all currently observable objects for which you can create threshold monitoring jobs. By default all objects of all observable classes and in every domain will be listed. The list of objects is sorted by their class name.

class=*objectClass*
Specifying the optional *objectClass* will restrict the output to observable objects of that specific class. The *objectClass* must be one of the classes listed by the `thrsh-job observable-classes` subcommand.

domain=*objectDomain*
Specifying the optional *objectDomain* will restrict the output to observable objects in that domain. The domain of an object is the string preceding the colon (":") character in the object's name.

`thrsh-job observable-attributes class=`*objectClass*
Display the list of all observable attributes in the specified *objectClass*. Attributes are displayed with their name and type. The *objectClass* must be one of the classes listed by the `thrsh-job observable-classes` subcommand.

`thrsh-job list`
Displays the list of all currently defined threshold monitoring jobs. Jobs are listed for each object having a defined threshold job, and objects are sorted by their class name. The information displayed for each job is the same as displayed by the `thrsh-job info` subcommand.

Security: For other users than the one who launched the Common Agent Container, no jobs will be displayed.

`thrsh-job info` *jobName*

Display verbose information about a threshold monitoring job named *jobName*. The *jobName* must be one displayed by the `thrsh-job list` subcommand. This subcommand displays the following information:

- The name of the threshold monitoring job.
- The multiplicity of the threshold monitoring job. In this release, only simple threshold jobs that monitor one attribute on one object are possible.
- The state of the threshold monitoring job: active on-duty, active off-duty, or suspended. An active on-duty job is currently scheduled to run and is collecting data. An active off-duty job is running but not collecting data because the current time is out of its

working schedule. A suspended job is not running nor collecting any data. Use the `thrsh-job suspend` and `thrsh-job resume` subcommands to change the running state of a threshold monitoring job.

- The granularity in seconds of the threshold monitoring job. This is the interval for data collection by this job.
- The schedule of the threshold monitoring job. The schedule specifies what days and times the job is active on-duty or active off-duty (collecting data or not, respectively).
- The alarm configuration of the threshold monitoring job. This is the alarm that will be triggered when the observed value of the monitored attribute crosses the defined threshold value. The display includes the alarm's type and severity.
- The observed object of the threshold monitoring job.
- The attribute name to which the threshold is applied.
- The value of the threshold that will trigger an alarm.
- The direction of the value's progress that will trigger an alarm at the threshold, either `RISING` or `FALLING`.
- The tolerance offset of the threshold. When the direction is `RISING`, an alarm will not be triggered again until the observed attribute is less than the *thresholdValue-offsetValue*. When the direction is `FALLING`, an alarm will not be triggered again until the observed attribute is more than the *thresholdValue+offsetValue*. This behavior holds true even when the offset is zero.

Security: For users other than the one who launched the Common Agent Container, no information will be displayed.

`thrsh-job create` *jobName* object=*objectName* granularity=*integerValue* attributeName=*attributeName* attributeType=*attributeType* thresholdValue=*thresholdValue* thresholdOffset=*offsetValue* thresholdDirection=`[RISING|FALLING]`

Creates a new threshold monitoring job that monitors one attribute on a single object. When creating threshold jobs, the following parameters can be set:

jobName

A string uniquely identifying the threshold monitoring job. The *jobName* cannot be already in use by any other threshold monitoring job.

object=*objectName*

The observable object on which the threshold monitoring job will collect the attribute values to compare against the threshold. The *objectName* must be one displayed by the `thrsh-job list` or `thrsh-job observable-objects` subcommands.

granularity=*integerValue*

The time specified in seconds between the initiation of two successive observations of the attribute value, while the job is active on-duty.

attributeName=*attributeName*
The name of the attribute for which the threshold monitoring job gathers values and compares compare them to the threshold. The *attributeName* must be listed by the `thrsh-job info` or `thrsh-job observable-attributes` subcommands.

attributeType=*attributeType*
The type of the observable attribute to be monitored. The *attributeType* must be listed by the `thrsh-job info` or `thrsh-job observable-attributes` subcommands.

thresholdValue=*thresholdValue*
The value of the monitored attribute that will cause this threshold job to trigger an alarm when crossed in the direction specified by `thresholdDirection`.

thresholdOffset=*offsetValue*
The *offsetValue* determines the tolerance of the threshold job in triggering successive alarms. The *offsetValue* must be zero or a positive value. After an alarm event is triggered, no new alarm event will be triggered until the value of the monitored attribute exceeds the range defined by the *offsetValue* and the `thresholdDirection`.

thresholdDirection=`[RISING|FALLING]`
When the direction is `RISING`, an alarm event will not be triggered again until the observed attribute value is less than *thresholdValue-offsetValue*. When the direction is `FALLING`, an alarm event will not be triggered again until the observed attribute value is more than *thresholdValue+offsetValue*. This behavior holds true even when the *offsetValue* is zero.

Security: This subcommand can be run only by the user who launched the Common Agent Container.

`thrsh-job delete` *jobName*
Delete a threshold monitoring job named *jobName*. The *jobName* must be one displayed by the `thrsh-job list` subcommand.

Security: This subcommand can be run only by the user who launched the Common Agent Container.

`thrsh-job suspend` *jobName*
Suspend a threshold monitoring job named *jobName*. A suspended job is not active and will no longer collect data, regardless of its schedule. However, the job remains defined and can be made active again with the `thrsh-job resume` subcommand. The *jobName* must be one displayed by the `thrsh-job list` subcommand.

Security: This subcommand can be run only by the user who launched the Common Agent Container.

`thrsh-job resume` *jobName*
: Resume a threshold monitoring job named *jobName*. A resumed job will begin collecting data and sending reports according to its schedule. The *jobName* must be one displayed by the `thrsh-job list` subcommand. This is the counterpart to the `thrsh-job suspend` subcommand.

  Security: This subcommand can be run only by the user who launched the Common Agent Container.

Examples

The following fictional scenario demonstrates how to use the `mfwkadm` utility, along with its options and subcommands.

The `list-modules` subcommand shows the Java ES component instances on the current host and their corresponding modules in the Common Agent Container. The following example lists two installed components, Directory Server having no running instances and Web Server having one running instance.

```
$ mfwkadm list-modules

Installed products and their running instances:
===============================================

Instances for installed product: com.sun.cmm.ds:collectionID=/opt/SUNWdsee/ds6,
name=Sun Java(TM) System Directory Server,type=CMM_InstalledProduct
------------------------------

No instance.

Instances for installed product: com.sun.cmm.ws:collectionID=/var/opt/SUNWwbsvr7,
name=WebServer,type=CMM_InstalledProduct
------------------------------

/wsPrefix/com.sun.cmm.ws:name=https-hostname.example.com,type=CMM_ApplicationSystem
```

The following `info` subcommand displays observable objects in the Web Server instance, with their classes and attributes for each job type.

```
$ mfwkadm info /wsPrefix/com.sun.cmm.ws:name=https-hostname.example.com,\\
type=CMM_ApplicationSystem

Information about running instance: /wsPrefix/com.sun.cmm.ws:
name=https-hostname.example.com,type=CMM_ApplicationSystem
==================================
```

```
Observable objects for performance jobs:
----------------------------------------

+ Objects of class: com.sun.cmm.settings.CMM_ApplicationSystemSetting

        /wsPrefix/com.sun.cmm.ws:name=https-hostname.example.com-setting,
type=CMM_ApplicationSystemSetting

        Observable attributes:

        Caption [STRING]
        ConfigurationDirectory [STRING]
        CreationClassName [STRING]
        Description [STRING]
        DirectoryName [STRING]
        ElementName [STRING]
        InstanceID [STRING]
        Name [STRING]
        URL [STRING]

+ Objects of class: com.sun.cmm.settings.CMM_KeepAliveSetting

        /wsPrefix/com.sun.cmm.ws:name=process-1-keepalive-setting,
type=CMM_KeepAliveSetting

        Observable attributes:

        AllocationUnit [STRING]
        Caption [STRING]
        ConnectionsUpperBound [LONG]
        CreationClassName [STRING]
        Description [STRING]
        ElementName [STRING]
        InputUnit [STRING]
        InstanceID [STRING]
        LowerAllocationLimit [LONG]
        LowerInputLimit [LONG]
        LowerOutputLimit [LONG]
        Name [STRING]
        OtherAllocationUnit [STRING]
        OtherInputUnit [STRING]
        OtherLowerAllocationLimit [LONG]
        OtherLowerInputLimit [LONG]
        OtherLowerOutputLimit [LONG]
        OtherOutputUnit [STRING]
        OtherUpperAllocationLimit [LONG]
        OtherUpperInputLimit [LONG]
        OtherUpperOutputLimit [LONG]
```

```
        OutputUnit [STRING]
        QueuedUpperBound [LONG]
        SecondsTimeout [LONG]
        TimeoutUpperBound [LONG]
        UpperAllocationLimit [LONG]
        UpperInputLimit [LONG]
        UpperOutputLimit [LONG]
        ...
```

The following command shows the list of defined performance monitoring jobs. In this example, there is one performance job called `myPerfJob` that monitors one object:

```
$ mfwkadm pm-job list

BY_OBJECTS performance jobs:
============================

Performance job information for: myPerfJob
------------------------------

Type:                BY_OBJECTS
State:               ACTIVE_ON_DUTY
Granularity period:  30
Reporting period:    1
By event:            EVENT_SINGLE
By file:             EVENT_SINGLE
Report format:       XML
Schedule:
        Global start time: Immediately
        Global stop time: Forever
        Weekly schedule: Everyday
        Daily schedule: All day
Observed objects:
                /wsPrefix/com.sun.cmm.ws:name=virtualServer-hostname.example.com-
webApp-/-stats,type=CMM_VirtualServerWebModuleStats
Observed attributes:
                All available

BY_CLASSES performance jobs:
============================

No jobs found.
```

The following command creates an operational status monitoring job related to two observable objects obtained from the `opstat-job info` or `opstat-job observable-objects` subcommands:

```
$ mfwkadm opstat-job create myOpStatJob granularity=60 \\
object=/wsPrefix/com.sun.cmm.ws:name=process-1,type=CMM_UnixProcess \\
```

```
object=/wsPrefix/com.sun.cmm.ws:name=process-1-DNSCache1,type=CMM_DnsCache
```

The following command suspends the job created above:

```
$ mfwkadm opstat-job suspend myOpStatJob
```

The following command shows the observable classes for the potential threshold monitoring jobs:

```
$ mfwkadm thrsh-job observable-classes

Threshold jobs observable classes:
==================================

com.sun.cmm.cim.CIM_ScopedSettingData
com.sun.cmm.cim.CIM_SettingData
com.sun.cmm.cim.CIM_StatisticalData
com.sun.cmm.cim.statistics.CIM_EthernetPortStatistics
com.sun.cmm.cim.statistics.CIM_NetworkPortStatistics
com.sun.cmm.cim.statistics.j2ee.CIM_J2eeJVMStats
com.sun.cmm.cim.statistics.j2ee.CIM_J2eeStatistic
com.sun.cmm.settings.CMM_ApplicationSystemSetting
com.sun.cmm.settings.CMM_KeepAliveSetting
com.sun.cmm.settings.CMM_QueueTimeoutSetting
com.sun.cmm.settings.CMM_RFC2788ApplicationSystemSetting
com.sun.cmm.settings.CMM_ScopedSettingData
com.sun.cmm.settings.CMM_SoftwareResourceSetting
com.sun.cmm.settings.CMM_SWRBufferSetting
com.sun.cmm.settings.CMM_SWRLimitSetting
com.sun.cmm.settings.CMM_SWRQueueSetting
com.sun.cmm.settings.CMM_VirtualServerSetting
com.sun.cmm.statistics.CMM_ApplicationSystemStats
com.sun.cmm.statistics.CMM_ApplicationSystemWatchdogStats
com.sun.cmm.statistics.CMM_ConnectionQueueStats
com.sun.cmm.statistics.CMM_DnsCacheStats
com.sun.cmm.statistics.CMM_EthernetPortStats
com.sun.cmm.statistics.CMM_FileCacheStats
com.sun.cmm.statistics.CMM_HTTPResponsesStats
com.sun.cmm.statistics.CMM_JVMJSR174ExtStats
com.sun.cmm.statistics.CMM_JVMJSR174Stats
com.sun.cmm.statistics.CMM_JVMStats
com.sun.cmm.statistics.CMM_NetworkPortStats
com.sun.cmm.statistics.CMM_OperatingSystemStats
com.sun.cmm.statistics.CMM_ProcessorStats
com.sun.cmm.statistics.CMM_ProcessStats
com.sun.cmm.statistics.CMM_QueueTimeoutStats
com.sun.cmm.statistics.CMM_RFC2788ApplicationTableStats
com.sun.cmm.statistics.CMM_ServiceStats
```

```
com.sun.cmm.statistics.CMM_SoftwareResourceStats
com.sun.cmm.statistics.CMM_SolarisEthernetPortStats
com.sun.cmm.statistics.CMM_SolarisNetworkPortStats
com.sun.cmm.statistics.CMM_SolarisOperatingSystemStats
com.sun.cmm.statistics.CMM_SolarisProcessorStats
com.sun.cmm.statistics.CMM_SolarisProcessorSysinfoStats
com.sun.cmm.statistics.CMM_SolarisProcessorVmStats
com.sun.cmm.statistics.CMM_Statistic
com.sun.cmm.statistics.CMM_SWRBufferStats
com.sun.cmm.statistics.CMM_SWRCacheStats
com.sun.cmm.statistics.CMM_SWRLimitStats
com.sun.cmm.statistics.CMM_SWRQueueStats
com.sun.cmm.statistics.CMM_UnixOperatingSystemStats
com.sun.cmm.statistics.CMM_UnixProcessStats
com.sun.cmm.statistics.CMM_VirtualServerWebModuleStats
com.sun.cmm.statistics.CMM_WebModuleStats
```

The following command shows the observable attributes for threshold jobs that monitor objects of class `com.sun.cmm.statistics.CMM_SWRQueueStats` found in the previous example:

```
$ mfwkadm thrsh-job observable-attributes \\
class=com.sun.cmm.statistics.CMM_SWRQueueStats

Threshold jobs observable attributes:
=====================================

Class: com.sun.cmm.statistics.CMM_SWRQueueStats

Attributes:

BufferSize [LONG]
EntriesCount [LONG]
EntriesHighWaterMark [LONG]
EntriesLowWaterMark [LONG]
EntriesTotal [LONG]
ErrorCount [INTEGER]
FailedOperations [LONG]
LowerLimit [LONG]
OperationsCount [LONG]
OtherLowerLimit [LONG]
OtherUpperLimit [LONG]
OverflowsCount [LONG]
QueuedCount [LONG]
QueuedHighWater [LONG]
SampleInterval [LONG]
TotalQueuedCount [LONG]
UpperLimit [LONG]
```

The following command is another example of job creation, here with a threshold job:

```
$ mfwkadm thrsh-job create myThreshJob granularity=30 \\
object=/wsPrefix/com.sun.cmm.ws:name=process-1-threadPool-NativePool-stats,\\
type=CMM_SWRQueueStats attributeName=EntriesCount attributeType=LONG \\
thresholdValue=1000 thresholdOffset=10 thresholdDirection=RISING
```

The following example demonstrates the output of the `thrsh-job info` subcommand for the threshold monitoring job created in the previous example:

```
$ mfwkadm thrsh-job info myThreshJob

Threshold job information for: myThreshJob
----------------------------

Type:               SIMPLE
State:              ACTIVE_ON_DUTY
Granularity period:  30
Schedule:
        Global start time: Immediately
        Global stop time: Forever
        Weekly schedule: Everyday
        Daily schedule: All day
Alarm configuration:
               Type: QualityOfServiceAlarm
               Severity: INDETERMINATE
Threshold definition(s):
               Object: /wsPrefix/com.sun.cmm.ws:name=process-1-threadPool-
NativePool-stats,type=CMM_SWRQueueStats
                     Attribute: EntriesCount [LONG]
                     Value: 1000
                     Direction: RISING
                     Offset: 10
```

Exit Status

The following exit values are returned:

0 Successful completion

1 An error occurred

Attributes

Attribute Type	Attribute Value
Availability	SUNWmfwk
Interface Stability	Contract Private

See Also

cacao.5, cacaoadm.1m

◆ ◆ ◆ CHAPTER 3

Installing and Using Monitoring Console

Monitoring Console is the web-based application that displays all of the monitoring data collected by the instrumentation. It relies on a master agent to aggregate all the values and alarm notifications from each of the node agents.

Once Monitoring Console is installed, you can access it securely from a simple browser window on any host, even over the Internet if your firewall is configured to allow it. Using the graphical interface, you can then see monitored values in real-time, view and acknowledge alarms, and create rules for triggering custom alarms.

Note – Before you perform any installation or configuration, you should consult the *Sun Java Enterprise System 5 Release Notes for UNIX*.

This chapter contains the following sections:

Installing the Monitoring Console

Due to limitations of the master agent in this release, you cannot have a master agent on the same host as a node agent. As a consequence, the Monitoring Console cannot be installed on the same host as any of the other monitored components of Java ES. It must be installed on its own host, unless you have configured Solaris zones. For more information, see below "To Install the Monitoring Console in a Solaris Zone" on page 51

The installation of Monitoring Console also installs Monitoring Framework as a shared component dependency. The console requires the framework and the Common Agent

Container to load the master agent, but unlike the node agent, the master agent is not user-configurable. Specifically, you should not use the `mfwkadm` command on the host or in the zone where you install Monitoring Console.

▼ To Install the Monitoring Console with the Java ES Installer

Because of a limitation in this beta release, you must install Monitoring Console on a host or Solaris zone where no other Java ES component is installed. As a result, the Monitoring Console is the only component you will install in this procedure.

This procedure uses the installer's graphical interface. For information on how to run the installer in other modes, see Chapter 4, "Installing With the Text-Based Interface," in *Sun Java Enterprise System 5 Installation Guide for UNIX* and Chapter 5, "Installing in Silent Mode," in *Sun Java Enterprise System 5 Installation Guide for UNIX*.

1. **Launch the `installer` application from the directory the corresponds to your platform in the Java ES release. For further details, see "To Begin Installation" in *Sun Java Enterprise System 5 Installation Guide for UNIX*.**
2. **After continuing past the welcome screen and accepting the license, choose to Upgrade or Install, select Install New Software, and then click next.**
3. **On the component selection screen, select only the Sun Java System Monitoring Console to be installed. Click Next.**
4. **The installer checks for needed upgrades to the shared components. When it is done, click Next.**
5. **The installer now checks system requirements. If your operating system needs patches, Cancel the installation, add the required patches to your system and restart this procedure. Otherwise click Next.**
6. **On the configuration type selection screen, chose Configure Now and on the next custom configuration screen, click Next.**
7. **The installer is ready to install the Monitoring Console, click Next to begin. During the installation, you may open the product registration window if you have not yet registered your deployment of Java ES.**
8. **When the installation is done, you may review the installation summary and logs and then click Installation Complete to exit the installer.**

Next Steps You should now proceed "To Configure the Monitoring Console" on page 51.

▼ To Install the Monitoring Console in a Solaris Zone

By using Solaris zones, you can install the Monitoring Console on the same physical host as other components of Java ES. Those components will be in the global zone, and you will create a sparse root local zone to be a logical host for the Monitoring Console. Proceed in the following order.

1. **Install and configure all of you Java ES components except the Monitoring Console in the global zone. Complete all post-installation configuration of your selected components in the global zone so that all server instances are running.**

2. **As part of the installation in the global zone, the Monitoring Framework will be installed as a shared component in the global zone. Perform all procedures in Chapter 2 that are applicable to your installed components.**

3. **On the same host, create a sparse root local zone as the logical host for the Monitoring Console. Because it is a sparse root zone, the Monitoring Framework installed in *mfwk-base* should be visible (see "Default Paths and File Names" on page 9).**

4. **Install the Monitoring Console in the sparse root local zone following the procedure "To Install the Monitoring Console with the Java ES Installer" on page 50.**

5. **Configure the Monitoring Framework in the sparse root zone with the following commands:**

   ```
   cd mfwk-base/bin
   ./mfwksetup -i
   ```

   Using the files from the global zone, this command will create the necessary Monitoring Framework configuration files in the local zone.

Next Steps You should now proceed "To Configure the Monitoring Console" on page 51.

▼ To Configure the Monitoring Console

This procedure describes how to configure Monitoring Console on a separate physical host. If you installed Monitoring Console on a logical host created by a Solaris zone, the commands are the same but they must be run within that zone's file system.

1. **Use the Monitoring Framework to initialize the master agent with the following commands:**

   ```
   cd mfwk-base/bin
   ./masetup -i
   ```

2. **Restart the Common Agent Container (`cacao`) with the following command:**

   ```
   cacaoadm restart
   ```

▼ To Unconfigure the Monitoring Console

If you install and configure Monitoring Console on a host where you wish to install other components, you will not be able to monitor those components because of a conflict in the Monitoring Framework. To monitor the new components with a node agent, you must unconfigure the master agent of the Monitoring Console.

◗ **As root, run the following commands to unconfigure the Monitoring Console:**

```
cacaoadm stop
cacaoadm unregister-module com.sun.mfwk.masteragent.xml
cacaoadm register-module /etc/mfwk-base/xml/com.sun.mfwk.xml
cacaoadm restart
```

Installed Directory Layout

For the name of the package installed on your operating system, see Chapter 5, "List of Installable Packages," in *Sun Java Enterprise System 5 Installation Reference for UNIX*. The following table describes the directories in the Monitoring Console package. The default installation directory *MConsole-base* has the following meaning, as described in "Default Paths and File Names" on page 9:

- Solaris systems: `/opt/SUNWjesmc`
- Linux systems: `/opt/sun/jesmc`

TABLE 3–1 Directories and Files Used by the Monitoring Console

Path	Description of contents
MConsole-base`/WEB-INF/classes`	Web application servlet classes
MConsole-base`/WEB-INF/lib`	Web application JAR dependencies
MConsole-base`/WEB-INF/*.xml`	Web application descriptors
MConsole-base`/css`	Stylesheet files
MConsole-base`/html`	HTML files
MConsole-base`/images`	GIF image files used in the user interface
MConsole-base`/js`	JavaSript™ files
MConsole-base`/*.jsp`	JavaServer Pages™ files
WebConsole-base`/prereg/jesmc/*.reg`	Web Console files for the Monitoring Console

Starting the Monitoring Console

The Monitoring Console is a web application available through any browser that can connect to the host where you installed it. You will access the Monitoring Console through the Web Console that is automatically installed on the same host. The following procedure explain how to access the Monitoring Console and view the monitored components.

▼ To Launch the Monitoring Console

1 **You must first restart the web server for the Web Console. Run this command on the host or in the zone where you have installed Monitoring Console:**

```
/usr/sbin/smcwebserver restart
```

2 **Wait for the Web Console to be started. Use the following command to see if it is ready:**

```
/usr/sbin/smcwebserver status
```

You might need to run this command several times until you see the following message:

```
Sun Java(TM) Web Console is running.
```

3 **Open the Web Console using the following URL from any browser that can connect to the Monitoring Console host. If you installed in a Solaris zone,** *MC-host* **is the logical hostname you gave to that zone:**

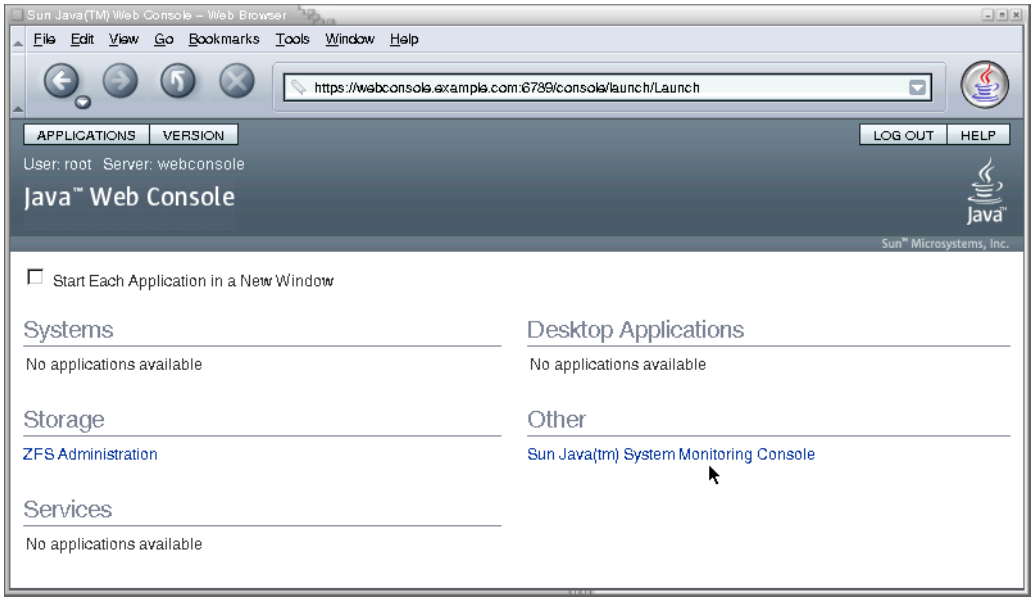
`https://`*MC-host*`.`*domain*`:6789`

4 **Depending on how your browser is configured, you may see a message about an untrusted certificate. You will need to trust the certificate to access the Web Console.**

5 **When prompted, login to the Web Console as** `root` **using the root password on the Monitoring Console host.**

When you are logged in, the Web Console lists all the services that it provides.

6 **To open the main window of the Monitoring Console, click on Sun Java System Monitoring Console under the heading titled "Other," as shown in the following screen capture.**



▼ To Connect to Your Node Agents

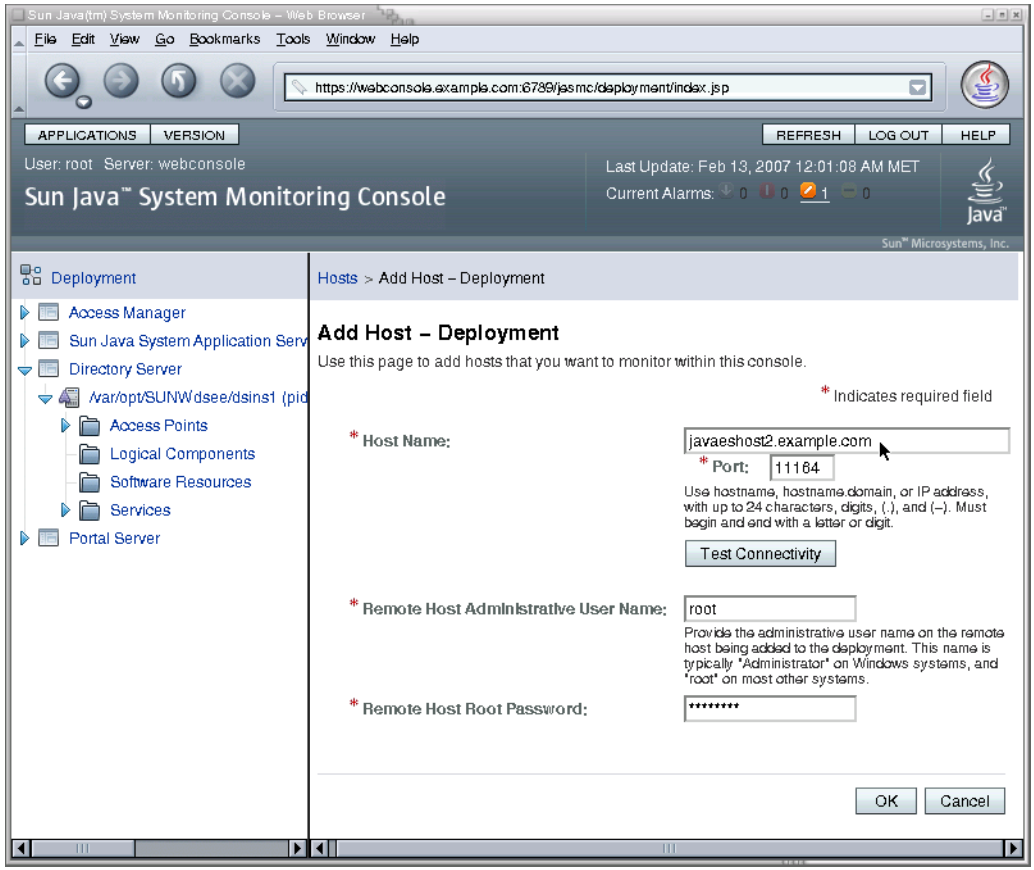
When the Monitoring Console is started for the first time, you must indicate where your monitored components are hosted. You specify the location of each node agent in your Java ES deployment, and the console will automatically display all of the components in each node agent. You will also need to repeat this procedure if you later add Java ES components to your deployment by installing them on new hosts.

Once you add a node agent, the Monitoring Console will reconnect to it every time you access the console until you remove it explicitly. If a node agent you previously added is not responding, follow the procedure "To Restart a Node Agent" on page 27.

1 **Synchronize the date and time between the logical host where the Monitoring Console is installed and the host containing the node agent and Java ES components you wish to monitor. Regardless of whether you synchronize automatically or manually, the time on each host must be within approximately 10 minutes of each other.**

2 **If necessary, navigate to the Deployment level display by clicking the "Deployment" link at the root of the hierarchy in the left-hand side of the Monitoring Console. Now select the Hosts tab in the right-hand pane and click Add.**

3 **In the Add Host dialog that appears, enter the required information as shown in the following screen capture:**



- Host Name: enter the fully qualified host name of a node agent where you have configured monitored components.
- Port: 11164, unless you have otherwise configured the Monitoring Framework on the host where the node agent resides.
- Remote Host Root Password: enter the root password for the system where the node agent resides.

4 **Click Test Connectivity.**

If the connection information is correct and your host agent is configured and running, the dialog will indicate that it is now connected.

5 **Click OK to finish the Add Host dialog, and the new name appears in the list of Hosts. All of the monitored components in that host's node agent now appear in the left column as well.**

6 **Repeat this procedure for every host in your Java ES deployment where monitored components are installed.**

Next Steps You can now browse the components listed in the left-hand column to view their operational status, the monitored attributes they expose, and any alarms they have triggered.

Using the Monitoring Console

The procedures in this section describe how to interact with the Monitoring Console.

▼ To Selectively Disable and Re-Enable Monitoring

The Java ES monitoring mechanisms are designed to be lightweight in order to not impact the performance of production systems. However, in certain cases it is desirable to stop collecting monitoring values so that the instrumentation has nearly zero impact on performance. The Monitoring Console provides a way to do this on a host-by-host basis, as described in the following procedure.

1 **If necessary, navigate to the Deployment level display by clicking the "Deployment" link at the root of the hierarchy in the left-hand side of the Monitoring Console. Then click on the Hosts tab in the right-hand pane.**

The table on the Hosts tab lists all of the hosts containing Java ES component that are monitored by the Monitoring Console.

2 **Use the checkboxes in the left-hand column of the table to select all hosts that you wish stop monitoring. Click Disable at the top of the table of Hosts.**

More Information

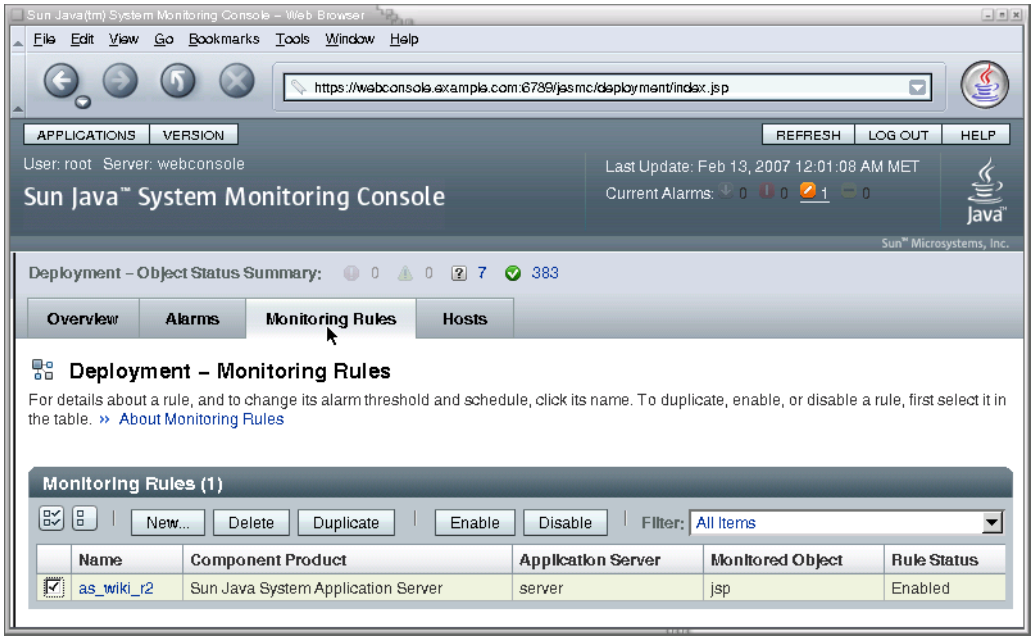
Consequences

When monitoring on a host is disabled, all monitoring object in the hierarchy under that host are disabled. In the disabled state, monitored objects are no longer updated, although they may still contain the last value. Monitoring rules that depend on a disabled object are suspended. To enable a host that was disabled, follow this procedure using the Enable button at the top of the table of Hosts.

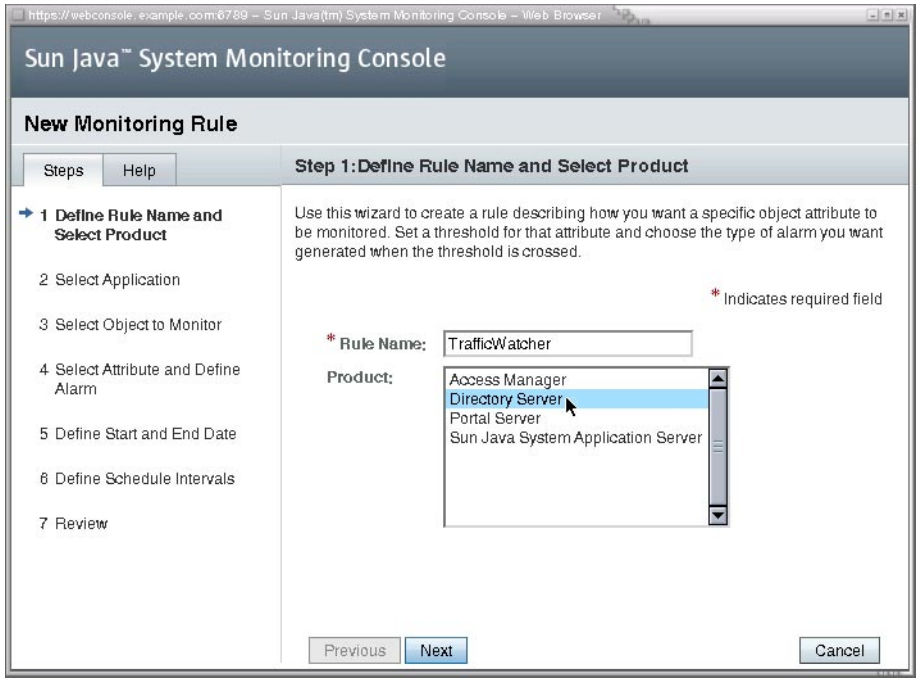
▼ To Create a New Monitoring Rule

A monitoring rule, also called a monitoring job, is a set of conditions of monitored values that the user defines to trigger an alarm. The monitoring rule wizard in the Monitoring Console helps you define the conditions you would like to monitor.

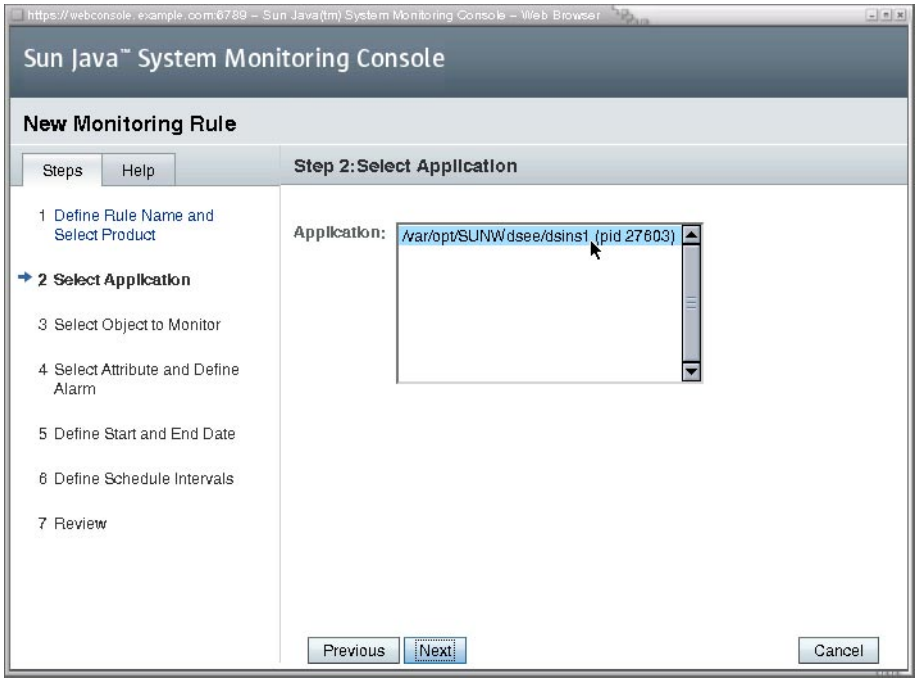
1 **If necessary, navigate to the Deployment level display by clicking the "Deployment" link at the root of the hierarchy in the left-hand side of the Monitoring Console. Now select the Rules tab in the right-hand pane as shown in the following screen capture, and click New in the table of monitoring rules:**



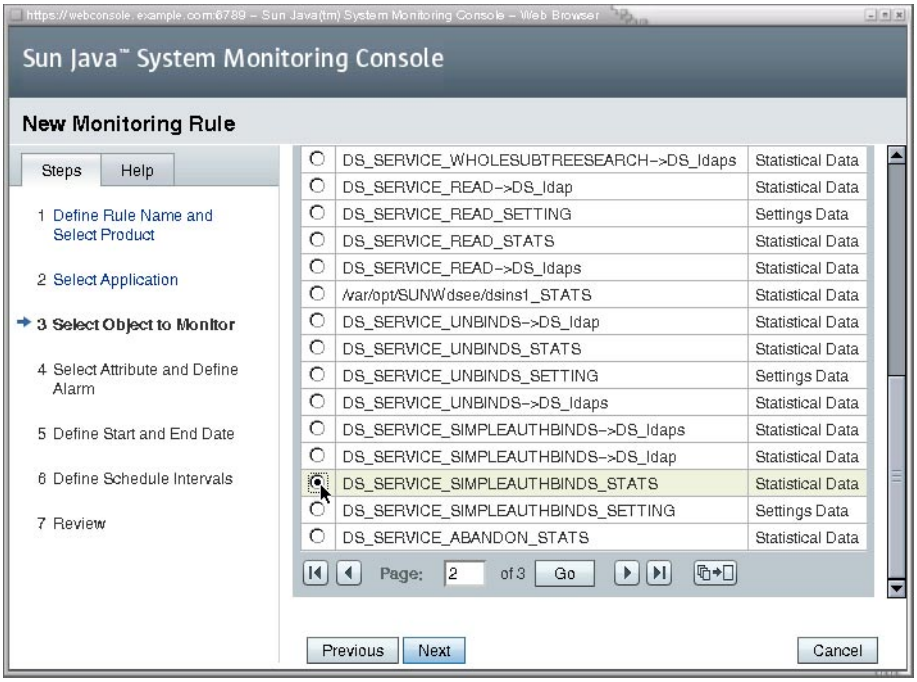
2 **Give a name to your new monitoring rule and choose the type of server you wish to monitor.**



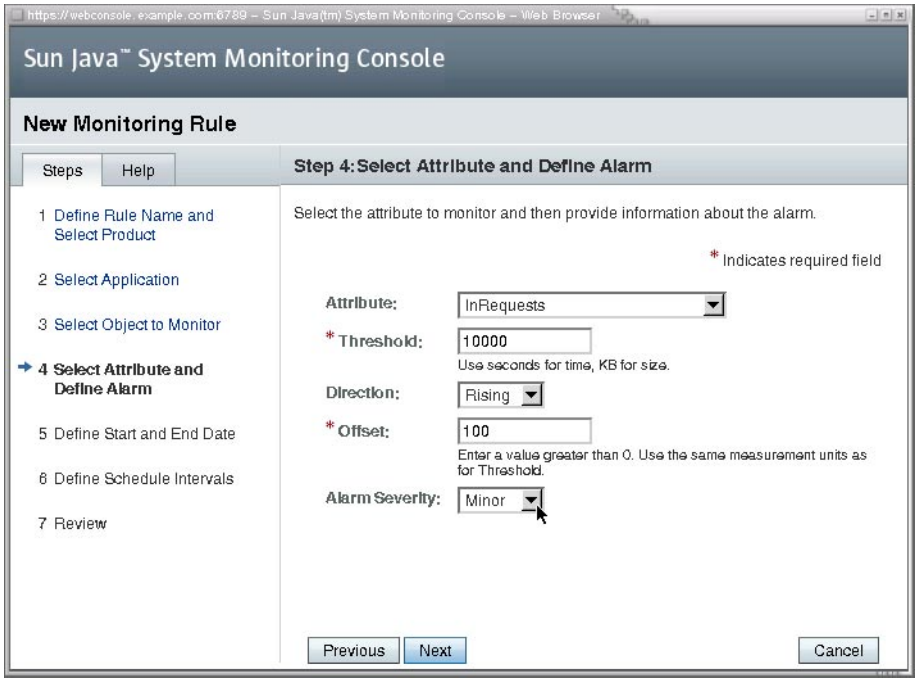
3 Choose the instance of the component product you wish to monitor. In the case when there are two instances of the same product installed on separate hosts, some instances may have identical names in this table. In this case, the instances may be in the same order as they appear in the hierarchy in the left-hand pane, but there is no sure way to know. You may need to create identical monitoring rules on both instances to ensure you rule is defined.



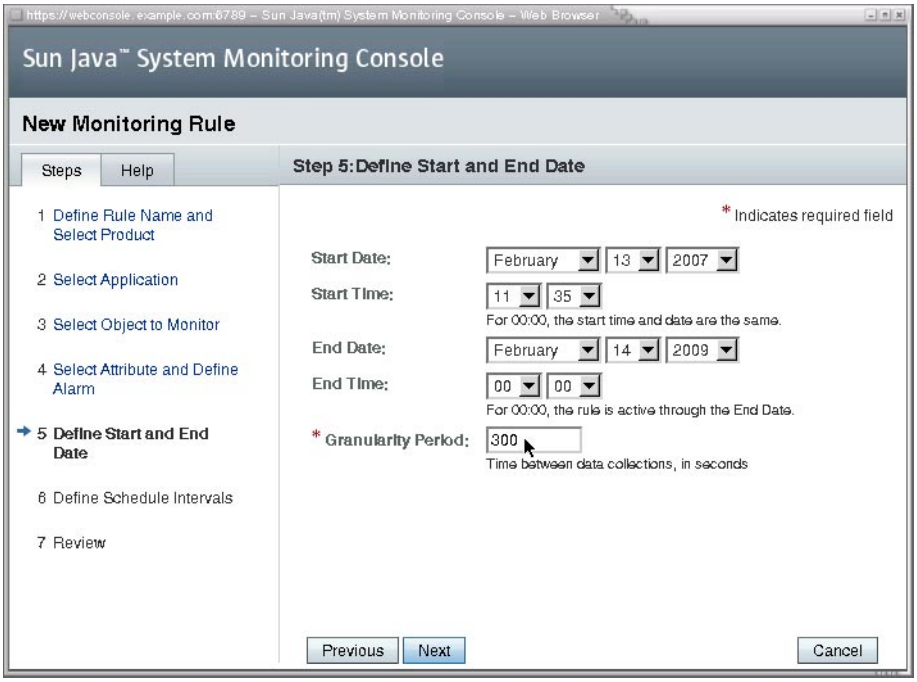
4 Select the object that contains the attribute you wish to monitor:



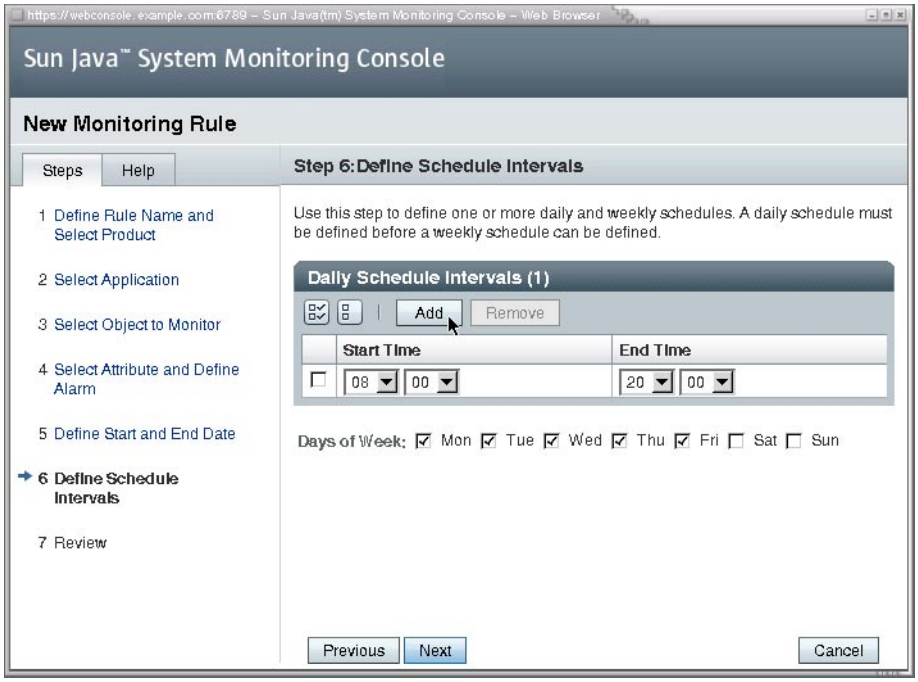
5 Now, you can finally specify the attribute that will be monitored along with the values that will generate an alarm.



6 **Enter start and end dates for the rule. Unlike the schedule which determines the activity or inactivity of a rule, the start and end dates determine when a rule is defined to exist. When a start time is in the past, as always happens with the default value, the monitoring associated with this rule will begin immediately**



7 Optionally, use the controls to create one or more time ranges during which the rule will be actively monitoring You can also select the days of the week to create a weekly schedule.



8 **In this final step of the Rule Wizard, review your input and click Finish to create your new rule.**

When the Rule Wizard is finished, you should see the Monitoring Rules tab again, with your new rule in the table of rules.

Troubleshooting the Monitoring Console

See also the known issues listed in the *Sun Java Enterprise System 5 Release Notes for UNIX*.

If the master agent is conflicting with a node agent, check the following conditions:

- If you use Solaris zones, make sure you installed Monitoring Console in a sparse root local zone.
- Make sure no previous installation of the monitored component remains on your host or in your zone.
- In either case, you will need to uninstall Monitoring Console and any components, correct the problem, and redo the Monitoring Console installation.

If you uninstall the Monitoring Console and reinstall it on the same host, it will fail to initialize and will not appear in the Web Console. In this case, run the `masetup –i` command on the Monitoring Console host to initialize the master agent. Then follow the procedure to "To Launch the Monitoring Console" on page 53.

Monitoring rules have a limitation that only allows them to be disabled when they are active. If you wish to disable a rule whose schedule makes it inactive at the current time, you must either change its schedule to make it active briefly, or remove the rule entirely.

Due to limitations the Windows platform, the `handleCount` and `threadCount` values in the host statistics are always 0 (zero).

◆ ◆ ◆ APPENDIX A

CMM Object Reference

The Common Monitoring Model (CMM) is an extension of the Common Information Model (CIM) implemented in the Java programming language. CIM is embodied in Java interfaces of the com.sun.cmm.cim.* packages. CMM is embodied in the interfaces of the `com.sun.cmm.*` packages that extend the CIM interfaces. Monitored objects are represented in the node agents by classes that implement the CMM interfaces. The following tables show what attributes can be monitored for each class of object.

Overview of CMM Objects

CMM is based upon a limited set of core interfaces that define which attributes a monitored object of that type can expose. The following list shows the classes that represent the broad types of monitored objects defined by CMM, and gives some of their key attributes:

`CMM_InstalledProduct`	A Java ES component product, taken as a whole. For example, Java ES Directory Server.
`CMM_ApplicationSystem`	An installed and configured instance of a Java ES component product. This instance can be either running or not running. Typical attributes of this object would be the contact information for the administrator, the operational status of the system, and startup or stopping time of the application.
`CMM_Service`	A specific function of a component product, for example, the Java ES Directory Server authentication service. A typical attribute would be the service's operational status.
`CMM_SoftwareResource`	A representation of software entities in the environment, such as a cache, thread pool, and so on. A typical attribute would be a cache size.

`CMM_LogicalComponent`	An entity which is manipulated by a Service and which is visible to the end user, but which does not represent an actual physical re source or a software feature. For example, a set of configuration parameters for a software instance, rather than the instance itself.
`CMM_ServiceAccessURI`	The point at which a Service is made available to be used. Typical attributes would be a port number or Uniform Resource Identifier (URI).
`CMM_RemoteServiceAccessPoint`	Access and addressing information for a remote connection. Typical attributes would be a URI, or the operational status of the connection (open or closed).
`CMM_Process`	A single instance of a running program. Typical attributes would be memory or CPU usage.
`CMM_UnitaryComputerSystem`	A single host used by the Java ES deployment, for example a desktop machine or a server. Typical attributes could be the number of available processors, or the amount of physical memory.
`CMM_OperatingSystem`	The software or firmware that makes a host machine's hardware usable. A typical attribute could be the amount of available virtual memory on the system.
`CMM_JVM`	The Java Virtual Machine used by a Java ES server. An example attribute could be the version number of the Java Virtual Machine.
`CMM_DatabaseService`	A task performed on behalf of a database, for example providing user access. A typical attribute could be the maximum permitted number of connections to the database.
`CMM_CommonDatabase`	Properties that are common across a given type of database. A typical attribute could be the date of the most recent back up.

◆ ◆ ◆ APPENDIX B

Monitored Objects Exposed by Each Component

The sections in this appendix list the CMM objects that have been instrumented in each product component that supports monitoring. When only a subset of an object's attributes are instrumented, those attributes are listed as well.

Instrumentation of the Common Agent Container

Not yet documented.

Instrumentation of Access Manager

Not yet documented.

Instrumentation of Application Server

Not yet documented.

Instrumentation of Calendar Server

Not yet documented.

Instrumentation of Directory Server

Not yet documented.

Instrumentation of Instant Messaging

Not yet documented.

Instrumentation of Messaging Server

Not yet documented.

Instrumentation of Portal Server

Not yet documented.

Instrumentation of Web Server

Not yet documented.

Index