Tivoli Management Framework

IBM

Maintenance and Troubleshooting Guide

Version 4.3.1

GC32-0807-02

Tivoli Management Framework

Maintenance and Troubleshooting Guide

Version 4.3.1

GC32-0807-02

Note

Before using this information and the product it supports, read the information in "Notices," on page 133.

Tivoli Management Framework Problem Determination Guide

Contents

Preface

This guide explains how to maintain the Tivoli® environment and troubleshoot systems errors and solve problems that can arise during normal operations.

Who should read this guide

This guide is intended for system administrators who are responsible for setting up and maintaining the Tivoli environment. To perform many of the operations described herein, you must have the super role for the Tivoli region or managed node being investigated. Many tasks also require you to have the UNIX® root account or Windows Administrator account on the system where Tivoli Management Framework is installed.

A working knowledge of Tivoli Management Framework and the overall Tivoli environment is necessary to understand the procedures outlined in these chapters. It is also essential that you understand the following:

- The architecture of the Tivoli environment as it is deployed for your enterprise
- The environment where Tivoli Management Framework is installed, such as the operating systems, hosts, and applications
- Concepts such as directories, files, and symbolic links
- Shell programming, such as the ksh shell, to run command examples from a Windows command line

Prerequisite and related documents

The following documentation is also useful for troubleshooting problems in the Tivoli environment:

- *Tivoli Management Framework Planning for Deployment Guide*

  Explains how to plan for deploying your Tivoli environment. It also describes Tivoli Management Framework and its services.
- *Tivoli Enterprise Installation Guide*

  Explains how to install and upgrade Tivoli Enterprise software within your Tivoli region using the available installation mechanisms provided by Tivoli Software Installation Service and Tivoli Management Framework. Tivoli Enterprise software includes the Tivoli server, managed nodes, gateways, endpoints, and RDBMS Interface Module (RIM) objects. This guide also provides information about troubleshooting installation problems.
- *Tivoli Management Framework User's Guide*

  Describes the concepts and procedures for using Tivoli Management Framework services. It provides instructions for performing tasks from the Tivoli desktop and from the command line.
- *Tivoli Management Framework Problem Determination Guide*

  Explains how to maintain a Tivoli environment and troubleshoot problems that can arise during normal operations.
- *Tivoli Management Framework Reference Manual*

Provides in-depth information about Tivoli Management Framework commands. This manual is helpful when writing scripts that are later run as Tivoli tasks. This manual also documents default and validation policy scripts used by Tivoli Management Framework.

Accessing publications online

The documentation CD contains the publications that are in the product library. The format of the publications is PDF, HTML, or both.

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli software information center Web site. Access the Tivoli software information center by first going to the Tivoli software library at the following Web address:

http://publib.boulder.ibm.com/tividd/td/tdprodlist.html

Note: If you print PDF documents on other than letter-sized paper, set the option in the **File → Print** window that allows Adobe Reader to print letter-sized pages on your local paper.

Ordering publications

You can order many Tivoli publications online at the following Web site:

http://www.elink.ibmlink.ibm.com

From this Web page, select **Publications** and follow the instructions.

You can also order by telephone by calling one of these numbers:
- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, see the following Web site for a list of telephone numbers:

http://www.ibm.com/software/tivoli/order-lit

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate all features of the graphical user interface.

Contacting software support

If you have a problem with any Tivoli product, refer to the following IBM Software Support Web site:

http://www.ibm.com/software/sysmgmt/products/support/

If you want to contact software support, see the *IBM Software Support Guide* at the following Web site:

http://techsupport.services.ibm.com/guides/handbook.html

The guide provides information about how to contact IBM Software Support, depending on the severity of your problem, and the following information:

- Registration and eligibility
- Telephone numbers, depending on the country in which you are located
- Information you must have before contacting IBM Software Support

Conventions used in this guide

This guide uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls
- Keywords and parameters in text

Italic

- Words defined in text
- Emphasis of words (words as words)
- New terms in text (except in a definition list)
- Variables and values you must provide

`Monospace`

- Examples and code examples
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

This guide uses the UNIX convention for specifying environment variables and for directory notation:

- When using the Windows command line, replace $*variable* with %*variable*% for environment variables and replace each forward slash (/) with a backslash (\) in directory paths.
- When using the bash shell on Windows operating systems, use the UNIX conventions.

Chapter 1. Maintaining the Tivoli environment

After you install and set up your Tivoli environment, it is important to maintain and tune your systems managed by Tivoli Management Framework for optimal performance.

This chapter explains how to perform the following tasks:
- "Setting Tivoli environment variables"
- "Placing a Tivoli region in maintenance mode" on page 2
- "Viewing data for a Tivoli region" on page 4
- "Listing active managed nodes" on page 7
- "Stopping and starting the object dispatcher" on page 8
- "Starting and stopping the endpoint service" on page 10
- "Communicating between managed nodes" on page 12
- "Determining if a managed node is connected" on page 12
- "Changing IP names and addresses" on page 16
- "Changing the Tivoli server configuration" on page 23
- "Improving performance in a Tivoli environment" on page 32
- "Deleting versioned system files" on page 30

Before beginning maintenance operations, make sure that you have the appropriate Tivoli administrator roles. For more information about authorization roles for administrators, see *Tivoli Management Framework Planning for Deployment Guide*.

Setting Tivoli environment variables

Before you can use the Tivoli desktop or commands, you must set up the Tivoli environment variables. You can manually run one of the scripts provided by Tivoli Management Framework or modify your initialization environment (UNIX operating systems only).

Setting environment variables for UNIX servers

For UNIX operating systems, the installation process creates the following setup scripts:

/etc/Tivoli/setup_env.csh

/etc/Tivoli/setup_env.sh

To set the Tivoli variables on a UNIX operating system, perform the following steps:
1. Log in to a UNIX Tivoli server or managed node either locally or using telnet.
2. For the Bourne (sh) or Korn (ksh) shells, enter the following command:
   ```
   . /etc/Tivoli/setup_env.sh
   ```
   For the C (csh) shell, enter the following command:
   ```
   source /etc/Tivoli/setup_env.csh
   ```

Optionally, you can change your login initialization procedure to use the appropriate setup file so that the necessary environment variables and search paths are automatically set when you log in to the Tivoli server or managed node.

For example, you can add the following to your initialization procedure:

For sh or ksh shells:
```
if [ -f /etc/Tivoli/setup_env.sh ]; then
   . /etc/Tivoli/setup_env.sh
fi
```

For the csh shell:
```
if ( -f /etc/Tivoli/setup_env.csh ) then
   source /etc/Tivoli/setup_env.csh
endif
```

Setting environment variables for Windows servers

For Windows operating systems, the installation process creates the following setup scripts:

- %SystemRoot%\system32\drivers\etc\Tivoli\setup_env.cmd
- %SystemRoot%\system32\drivers\etc\Tivoli\setup_env.sh

To set the Tivoli variables on a Windows operating system, perform the following steps:

1. Log in to a Windows Tivoli server or managed node.
2. From a DOS command prompt, enter:
   ```
   %SystemRoot%\system32\drivers\etc\Tivoli\setup_env
   bash
   ```
   The **bash** command starts the bash shell, which is a variation of the Bourne shell.

Placing a Tivoli region in maintenance mode

You can place the local Tivoli region in maintenance mode to immediately terminate all active Tivoli processes. In this mode, you can perform various maintenance and diagnostic tasks, including performing a backup of the entire Tivoli region.

When you place a Tivoli region in maintenance mode, you become the only authorized Tivoli administrator within the Tivoli region. The desktops of any other active Tivoli administrators are locked until the Tivoli region is returned to normal mode.

Notes:

1. Automated processes, such as distributed monitors and endpoint logins, still run while the Tivoli region is in maintenance mode.
2. You must be logged in to the Tivoli server to put the Tivoli region in maintenance mode.

The following table provides the context and authorization role required for this task.

Activity	**Context**	**Required Role**
Place the Tivoli region into maintenance mode	Tivoli region	super

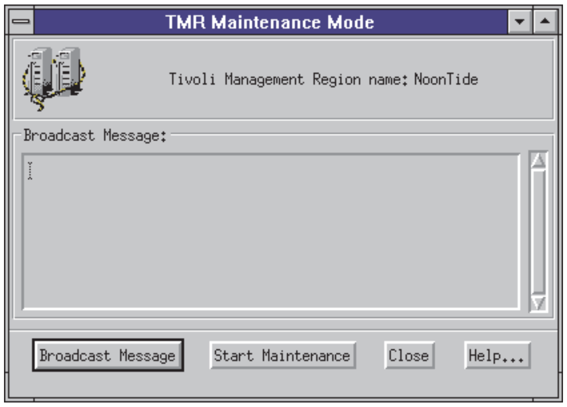
You can put a Tivoli region in maintenance mode from either the Tivoli desktop or command line.

Before you put the Tivoli region in maintenance mode, you can send a broadcast message to all active administrators in the Tivoli region.

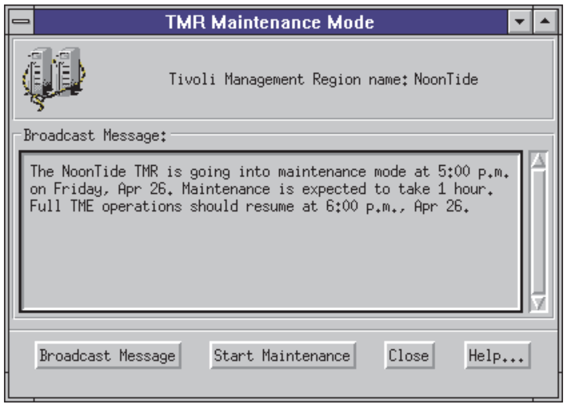
Desktop

To put a Tivoli region in maintenance mode, follow these steps:

1. Select **Maintenance** from the **Desktop** menu to display the TMR Maintenance Mode window.

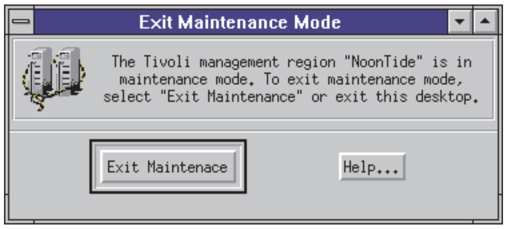


2. Enter a descriptive message in the **Broadcast Message** field.



```
The NoonTide TMR is going into maintenance mode at 5:00 p.m.
on Friday, Apr 26. Maintenance is expected to take 1 hour.
Full TME operations should resume at 6:00 p.m., Apr 26.
```

3. Click **Broadcast Message**. Tivoli Management Framework displays the message on the desktops of all active administrators in the Tivoli region. The message serves the same purpose in the Tivoli environment as the **shutdown** command message in the UNIX environment.
4. Click **Start Maintenance** to put the Tivoli region in maintenance mode.



The Tivoli management region "NoonTide" is in maintenance mode. To exit maintenance mode, select "Exit Maintenance" or exit this desktop.

The Tivoli region is now in maintenance mode. During this time, you are the only authorized Tivoli administrator able to perform management operations. The Exit Maintenance Mode window is displayed while the Tivoli region is in maintenance mode. To return the Tivoli region to normal operational status, click **Exit Maintenance** or exit the Tivoli desktop.

If the Tivoli desktop cannot exit maintenance mode, follow these steps:

1. To view the processes running on the Tivoli server, do one of the following:
   - On UNIX operating systems, enter one of the following commands:
     ```
     ps -aux
     ps -elf
     ```
   - On Windows operating systems, enter the following command:
     ```
     ntprocinfo
     ```
2. Determine which process is causing interference when attempting to exit maintenance mode. Note that processes you recognize are usually not the ones causing problems. In addition to looking for unusual processes, you can use the **grep** command to search the list of processes by date to see if the date of a suspect process coincides with when you placed the system in maintenance mode.
3. To stop the suspected process, do one of the following:
   - On UNIX systems only, enter the following command:
     ```
     kill pid
     ```
     where *pid* is the identifier for the process that you want to stop.
   - On Windows systems only, enter the following command:
     ```
     ntproc -k pid
     ```
     where *pid* is the identifier for the process that you want to stop.
4. Try to exit maintenance mode again.

Command line

For information about using the command line to put the current Tivoli region in maintenance mode, see the **wlocktmr** command in *Tivoli Management Framework Reference Manual*.

Viewing data for a Tivoli region

Before performing troubleshooting or maintenance operations, use the **odadmin** command to get a complete description of the local Tivoli region.

The following table provides the context and authorization role required for this task.

Activity	Context	Required role
View Tivoli region data	Tivoli region	**super**, **senior**, **admin**, or **user**

To view data for a Tivoli region, do one of the following:

- For data on the local object dispatcher, enter the following command:
  ```
  odadmin
  ```
- To view the status and configuration options of any managed node in the local Tivoli region, enter the following command:
  ```
  odadmin odinfo num
  ```

where *num* is the dispatcher number for the managed node.

For instructions on how to determine a dispatcher number, see "Listing active managed nodes" on page 7.

Output similar to the following is displayed:

```
1  Region = 1854004653
2  Dispatcher = 1
3  Interpreter type = w32-ix86
4  Database directory = C:\Tivoli\db\cherry.db
5  Install directory = C:\Tivoli\bin
6  Inter-dispatcher encryption level = simple
7  Kerberos in use = FALSE
8  Remote client login allowed = TRUE
9  Force socket bind to a single address = FALSE
10 Perform local hostname lookup for IOM connections = FALSE
11 Use Single Port BDT = FALSE
12 Use communication channel check = FALSE
13 Communication check timeout = default (180 secs)
14 Communication check response timeout = default (180 secs)
15 Oserv connection validation timeout = 03
16 Port range = (not restricted)
17 Single Port BDT service port number = default (9401)
18 Network security = none / SSL capable
19 SSL Ciphers = default
20 ALLOW_NAT = FALSE
21 State flags in use = TRUE
22 State checking in use = TRUE
23 State checking every 180 seconds
24 Dynamic IP addressing allowed = FALSE
25 Transaction manager will retry messages 4 times.
```

This output contains the following information:

Line 1 Indicates the region number, which is a unique number encoded within the license key.

Line 2 Indicates the server or object dispatcher number within the Tivoli region. Object dispatcher number 1 indicates a Tivoli server. The object dispatcher number is based on the installation order of the managed nodes.

Line 3
Indicates the machine interpreter type.

Lines 4 and 5
Identifies the path for the local Tivoli object database and the location of the binaries.

Line 6
Indicates the type of encryption that is used when managed nodes pass messages between themselves.

Line 7
Indicates whether Kerberos is being used within the Tivoli region.

Line 8 Indicates whether you can make a remote desktop connection using Tivoli Desktop for Windows.

Line 9 Indicates whether to force socket bind to a single address. For example, if this statement is FALSE and you have multiple network interface cards (NICs), the object dispatcher opens port 94 and binds to all IP addresses, one for each NIC. (TCP/IP allows the object dispatcher to listen on port 94 on all IP addresses.) If this statement is TRUE, it indicates that the object dispatcher binds to only port 94 on one IP address.

Line 10
Indicates that Inter-ORB Messaging (IOM) will use the IP address passed to make a connection back to the initiator of the IOM request. It will not use the host name passed to look up the IP address.

Line 11
Indicates whether single-port Bulk Data Transfer (BDT) is enabled (`TRUE`) or disabled (`FALSE`) for this node.

Line 12
Indicates whether a communication channel check is set to determine whether a network connection is still active

Line 13
Indicates the number of seconds after which a Tivoli process sends a ping message to determine whether an idle IPC channel has been terminated.

Line 14
Indicates the number of seconds after which a Tivoli process determines that an IPC channel is terminated.

Line 15
Lists the number of seconds that the object dispatcher attempts to validate an incoming connection. If this value is set to zero, incoming connection attempts do not time out.

Line 16
Identifies the range of ports that the Tivoli environment is allowed to use.

Line 17
Indicates the port that the BDT service uses on this node.

Line 18
Indicates the network security level of this node.

Line 19
Indicates the cipher list (in order of preference) used with SSL network security.

Line 20
Indicates whether network address translation (NAT) support is enabled (TRUE) or disabled (FALSE).

Line 21
Indicates whether the ping cache of the object dispatcher is consulted. For information about managed node pings, see "Determining if a managed node is connected" on page 12. This line applies to Tivoli servers only.

Line 22
Indicates whether the object dispatcher of the Tivoli server pings managed nodes to collect state information (TRUE) or whether it collects state information implicitly (FALSE). For more information, see "Determining if a managed node is connected" on page 12. This line applies to Tivoli servers only.

Line 23
Indicates the interval between pings. This line applies to Tivoli servers only.

Line 24
Indicates whether Dynamic Host Configuration Protocol (DHCP) support on managed nodes is enabled. This line applies to Tivoli servers only.

Line 25
Indicates the number of inter-ORB retries for communicating with another object dispatcher. This line applies to Tivoli servers only.

Note: On UNIX operating systems, the **odadmin** command also displays the library path in effect for Tivoli operations. On Windows operating systems, the dynamically linked libraries (DLLs) are stored in the binary directory instead of a separate library directory.

For information about how to set these configuration options, see the **odadmin** and **oserv** commands in the *Tivoli Management Framework Reference Manual*.

Listing active managed nodes

The object dispatcher service is the name of the Common Object Request Broker Architecture (CORBA)-compliant object request broker used by the Tivoli environment. It runs on the Tivoli server and each of the managed nodes connected to the Tivoli region. Each managed node communicates with the object dispatcher on the Tivoli server through a TCP/IP connection. The object dispatcher on the Tivoli server maintains a cached set of data based on its last communication to other object dispatchers.

This section describes how to use this service to list active (or connected) managed nodes and their status, any known machine name aliases, and IP addresses.

The following table provides the context and authorization role required for this task.

Activity	Context	Required role
List active managed nodes	Tivoli region	**super** or **senior**

To display the active managed nodes in a Tivoli region, enter the following command:

```
odadmin odlist
```

The **odlist** option lists host names, which are not always the same as the managed node names. If you have an entry for a given IP address in the local hosts file, that name is used.

Output similar to the following is displayed:

```
Region     Disp Flags Port IPaddr      Hostname(s)
1335980593 1    ct-   94   127.64.1.1  odin,odin.noontide.com
           3    ct-   94   127.64.1.2  balder,balder.noontide.com
           4    ct-   94   127.64.1.3  frey,frey.noontide.com
           5    ct-   94   127.64.1.4  loki,loki.noontide.com
           6    ct-   94   127.64.1.5  thor,thor.noontide.com
```

This output contains the following information:

Region
Specifies the unique Tivoli region identification number. This number identifies the installation in which the object dispatcher is located.

Disp Identifies a unique object dispatcher number. The object dispatcher numbers are incremented based on the next object to be created. Object dispatcher number 1 indicates a Tivoli server. Object dispatcher numbers are not reused.

Flags Specifies the following three fields:

- The first field indicates the connection status. The options are as follows:

  dash (–)
  Indicates that the connection is down.

  c Indicates an active connection.

  ? Indicates that the connection status is unknown.

  A lack of activity between the Tivoli server and the managed node occasionally results in the connection status indicating unknown when, in fact, the object dispatcher is up and running. This status is corrected when you **wping** the object dispatcher or perform an operation on the managed node. For more information, see "Communicating between managed nodes" on page 12.
- The second field indicates the trusted status of the connection. The options are as follows:

  t Indicates that the connection is trusted.

  – Indicates that the connection is not trusted.
- The third field is reserved for future use and is shown as a dash (–).

Port Indicates the port number. Port 94 is reserved for objcall/udp and objcall/tcp communications with the Network Information Center.

IPaddr
Indicates all known IP addresses for each managed node.

Hostname(s)
Indicates all host names for each managed node.

Stopping and starting the object dispatcher

Normally, you never need to start or stop the object dispatcher; the operating system manages its initiation and shutdown. However, there are times when you might need to manually halt operations or restart the object dispatcher. For example, situations in which you need to restart the object dispatcher include:

- Changing file system mount points
- Upgrading Tivoli Management Framework
- Installing a patch for the object dispatcher

In these cases, use the **odadmin** command to manually stop and start the object dispatcher service on managed nodes in the Tivoli region.

Note: To remotely start the object dispatcher on Windows operating systems, Tivoli Remote Execution Service must be installed.

The following procedures show you how to stop and start the Tivoli object dispatcher on various platforms. Before you perform these procedures, ensure that you have set the appropriate Tivoli environment variables. For more information, see "Setting Tivoli environment variables" on page 1. In addition, for Windows operating systems, ensure that Tivoli Remote Execution Service is running on all managed nodes running Windows operating systems.

The following table provides the context and authorization role required for this task.

Activity	Context	Required role
Start or stop the object dispatcher service	Tivoli region	**super** or **senior**

You can start or stop the object dispatcher from the command line only. For more information, see the *Tivoli Management Framework Reference Manual*.

Starting the object dispatcher

To start the object dispatcher on the local managed node, enter the following command:

```
odadmin start
```

To start the object dispatcher on a remote managed node, enter the following command where *num* is the number for the object dispatcher that you want to start:

```
odadmin start num
```

You can also use the following operating system-specific commands to start the object dispatcher.

- On a UNIX operating system, enter the following command:
  ```
  /etc/Tivoli/oserv.rc start
  ```
- On a Windows operating system, enter the following command:
  ```
  net start oserv
  ```
  You also can start the object dispatcher from the desktop of the Windows operating system using the following procedure:
  1. From the taskbar, select **Start → Settings → Control Panel** to open the Control Panel window.
  2. Double-click the **Services** icon to open the Services window.
  3. Scroll down the list of services and select **Tivoli Object Dispatcher**.
  4. Click **Start** to start the object dispatcher.

To start the object dispatcher on all managed nodes in the local Tivoli region, enter the following command:

```
odadmin start clients
```

To start the object dispatcher on the Tivoli server and its managed nodes, enter the following command:

```
odadmin start all
```

Restarting the object dispatcher

To restart the object dispatcher on a managed node, enter the following command:

```
odadmin reexec num
```

where *num* is an optional dispatcher number.

To restart the object dispatcher on all managed nodes in the local Tivoli region, enter the following command:

```
odadmin reexec clients
```

To restart the object dispatcher on the Tivoli server and its managed nodes, enter the following command:

```
odadmin reexec all
```

Stopping the object dispatcher

To shut down all managed nodes in the local Tivoli region, enter the following command:

```
odadmin shutdown clients
```

Note: the Tivoli server remains active.

To shut down the Tivoli server and all managed nodes in the Tivoli region, enter the following command:

```
odadmin shutdown all
```

The object dispatchers on the managed nodes are shut down in reverse order of their dispatcher number, from the highest to the lowest.

NetWare systems

You cannot start or restart the object dispatcher on NetWare gateways similar to other managed nodes. To start a NetWare system, enter one of the following commands:

```
odadmin reexec num
odadmin reexec clients
```

where *num* is an optional dispatcher number.

If you stopped the object dispatcher and need to start it, enter the following command:

```
SYS:TIVOLI\BIN\NWR-IX86\BIN\OSERVRUN
```

To run this command on a machine running the Windows operating system, use the NetWare **RCONSOLE** utility to start a remote console session. This utility is located in the SYS:PUBLIC directory on your NetWare system. Consult your NetWare documentation for more information about using this utility.

Starting and stopping the endpoint service

The endpoint service, or lcfd daemon, enables an endpoint to communicate with its assigned gateway. This service is similar to the object dispatcher service, which enables communication between the Tivoli server and its managed nodes.

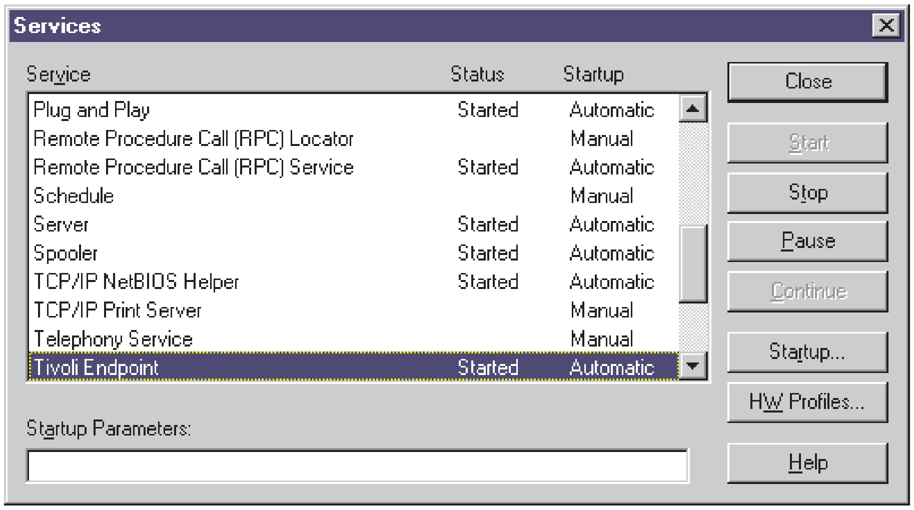
After the endpoint connects to its assigned gateway, the gateway address, port number, and any network aliases for the assigned gateway and alternate gateways are written to the lcf.dat file. All other configuration information is written to the last.cfg file. On subsequent startups, the startup commands (**lcfd** or **lcfd.sh**) read the configuration information from the lcf.dat file and the last.cfg file. As with the initial login, after the endpoint and gateway are connected, the configuration information is written to the last.cfg file.

If you start an endpoint using either the **lcfd** or **lcfd.sh** command, the options you specify override the equivalent entries in last.cfg file. The endpoint restarts with the new configuration, which is written to the last.cfg file when the connection is complete. This new configuration is used in all future startups.

The following procedures describe how to start or stop the endpoint service on your particular endpoint operating system.

Windows, except Windows 98
You can start and stop an endpoint from the Windows desktop or from the command line.

From the Windows desktop, click **Control Panel → Services**, and then start or stop the **Tivoli Endpoint** service.



Service	Status	Startup
Plug and Play	Started	Automatic
Remote Procedure Call (RPC) Locator		Manual
Remote Procedure Call (RPC) Service	Started	Automatic
Schedule		Manual
Server	Started	Automatic
Spooler	Started	Automatic
TCP/IP NetBIOS Helper	Started	Automatic
TCP/IP Print Server		Manual
Telephony Service		Manual
Tivoli Endpoint	Started	Automatic

From the command line, use the **net start lcfd** or **net stop lcfd** command to start or stop Windows endpoints.

Windows 98
If installed, double-click the endpoint icon in the Tivoli program group to start an endpoint.

Use the **lcfd –r** command to stop the endpoint.

UNIX Use the **lcfd.sh start** or **lcfd.sh stop** commands to start or stop UNIX endpoints.

NetWare
To start an endpoint from the NetWare console, use the **lcf** command. To stop an endpoint from the NetWare console, use the **lcfstop** command.

OS/2 Use the **start lcfd** command to start OS/2 endpoints or the **wos2proc** command to stop OS/2 endpoints.

To change the configuration of an endpoint, either edit the last.cfg file or restart the endpoint using one of the startup commands (**lcfd** or **lcfd.sh**) with the appropriate options. For information about these commands, see the *Tivoli Management Framework Reference Manual*. If you choose to edit the last.cfg file, the new configuration information is used when you restart the endpoint. When connected, the information is again written to the last.cfg file.

Communicating between managed nodes

The Tivoli environment is a distributed environment on top of which system management applications run. This environment consists of one or more machines that perform operations in a distributed and parallel fashion. Each managing system in a Tivoli region has a long-running object dispatcher that communicates with other Tivoli services on other machines. An operation initiated on one machine can start multiple operations on machines across the network, all running in parallel to complete their portion of the overall task.

The distributed architecture in a Tivoli environment is designed to work across a wide variety of Local Area Networks (LANs), Wide Area Networks (WANs), and network topologies. The minimal requirement is for bidirectional, full-time, interactive TCP/IP connections. In general, if you have Network File System (NFS) ability between two points on your network, you should have no problems running Tivoli Management Framework across those same two points.

The configuration of Tivoli regions and the location of file servers have a significant impact on the performance of a Tivoli environment. For example, if two sites are connected through a slow line over which requests and operations are run, you should make sure that each site is a Tivoli region and that it has a local file server with the appropriate Tivoli binaries. In this manner, the only traffic that passes over the slow line between the sites are management requests, not large amounts of data or requests from a remote Tivoli server.

Due to this distributed architecture, it is important that the communications and network function efficiently. To speed up error and timeout scenarios and to ensure reliable and accurate error handling and recovery, the Tivoli server tracks down machines that are temporarily unavailable due to network problems. You control whether the Tivoli server performs this tracking and, if so, how frequently it contacts the services on remote managed nodes.

Note: Keep in mind that the object dispatcher enables communication with managed nodes—not with endpoints. For an endpoint to communicate, it must start the lcfd service and communicate through its gateway. In turn, the endpoint manager on the Tivoli server communicates with the gateway using the object dispatcher process.

Determining if a managed node is connected

The object dispatcher on the Tivoli server occasionally pings connections to other Tivoli services to determine if a managed node is still connected. This information is maintained in a cached table of machines that have recently communicated with the Tivoli server.

The object dispatcher does not broadcast pings to all managed nodes in the Tivoli region. It only broadcasts pings if Tivoli operations are being performed, and only when all the following conditions are true:

- The client service is located in the local Tivoli region
- The client service has not sent a request to the Tivoli server during the most recent timeout period
- Another client service has sent a request for the client service during the most recent timeout period

Note: This algorithm minimizes the amount of network resources used.

Under some conditions, you might want to disable this automatic pinging. For example, if a connection is made over an automatic dial-up modem, it is undesirable to have the modem redial every 3 or 4 minutes just to handle ping messages. When a connection is over a channel that charges for individual packets, the overhead of occasional timeouts might be preferable to the cost of transmitting ping messages.

Another scenario involves enabling the use of the cached machine state and disabling polling that keeps the cache up to date. Under these circumstances, any timeouts are detected quickly and operations run at normal speeds. However, because polling is disabled, if a connection goes down, you need to manually exercise that connection to update the cache.

This section describes how to query and modify the object dispatcher ping behavior from the command line. The following table provides the context and authorization role required for these tasks.

Activity	Context	Required role
View and modify object dispatcher ping behavior	Tivoli region	**super** or **senior**

For more information, see the **odadmin** command in *Tivoli Management Framework Reference Manual*. For information about creating administrators with authorization roles for the Tivoli region, see the *Tivoli Management Framework User's Guide*.

Viewing the ping behavior of the object dispatcher

To display the current behavior of an object dispatcher ping, such as the interval between pings, follow these steps:

1. Log in to a managed node or the Tivoli server on which your Tivoli administrator has an alias with the **super** or **senior** role for the Tivoli region.
2. Ensure that your system environment is set correctly. For more information, see "Setting Tivoli environment variables" on page 1.
3. Enter the following command where *num* is the object dispatcher number:

   ```
   odadmin odinfo num
   ```

   Note: See "Viewing data for a Tivoli region" on page 4 for a line-by-line description of the **odadmin odlist** command output.

Enabling or disabling the ping cache of the object dispatcher

The following procedure enables the use of the ping cache of the object dispatcher. When the ping cache is enabled, subsequent management operations are attempted only if the entry in the ping cache indicates that the connection to the appropriate machine is up. If the connection indicates a down state, the operation is not attempted. The object dispatcher ping cache is, by default, enabled to prevent operations from timing out after 2 to 3 minutes, based on the return results of a typical TCP operation. (Some networks might take longer to time out.)

To enable the use of the ping cache of the object dispatcher, follow these steps:

1. Log in to a managed node or the Tivoli server on which your Tivoli administrator has an alias with the **super** or **senior** role for the Tivoli region.
2. Ensure that your system environment is set correctly. For more information, see "Setting Tivoli environment variables" on page 1.

3. Enter the following command:
   ```
   odadmin set_keep_alive on
   ```

The following procedure disables the use of the ping cache of the object dispatcher. When the ping cache is disabled, subsequent management operations are always attempted. If the connection is down, the operation eventually times out (usually within 2 to 3 minutes) when TCP/IP returns an error.

To disable the use of the object dispatcher ping cache, follow these steps:
1. Log in to a managed node or the Tivoli server on which your Tivoli administrator has an alias with the **super** or **senior** role for the Tivoli region.
2. Ensure that your system environment is set correctly. For more information, see "Setting Tivoli environment variables" on page 1.
3. Enter the following command:
   ```
   odadmin set_keep_alive off
   ```

Enabling and disabling pings for the object dispatcher

When object dispatcher is configured to ping managed nodes, those managed nodes that meet the conditions stated in "Determining if a managed node is connected" on page 12 are periodically polled to keep the object dispatcher ping cache accurate and up-to-date. The ping cache of the object dispatcher and object dispatcher pings to managed nodes are enabled by default. If this has changed and the object dispatcher ping is disabled, perform the following procedure to enable object dispatcher pings:
1. Log in to a managed node or the Tivoli server on which your Tivoli administrator has an alias with the **super** or **senior** role for the Tivoli region.
2. Ensure that your system environment is set correctly. For more information, see "Setting Tivoli environment variables" on page 1.
3. Ensure that the object dispatcher ping change is enabled. For instructions, see "Enabling or disabling the ping cache of the object dispatcher" on page 13.
4. Enter the following command:
   ```
   odadmin set_keep_alive poll
   ```

When you disable object dispatcher pings of managed nodes, the object dispatcher ping cache is updated only with information collected through implicit messages that are already being sent over the network. This reduces network traffic, but the object dispatcher might occasionally report a managed node as unavailable when it is available.

Note: If a managed node is available but is reported as unavailable because pinging is disabled, you should have that managed node send a request. A simple method is to run the **odadmin odinfo** command on the managed node that is reported as unavailable. Running this command on a different system does not correct this problem.

To disable object dispatcher pings to managed nodes, follow these steps:
1. Log in to a managed node or the Tivoli server on which your Tivoli administrator has an alias with the **super** or **senior** role for the Tivoli region.
2. Ensure that your system environment is set correctly. For more information, see "Setting Tivoli environment variables" on page 1.
3. Enter the following command:
   ```
   odadmin set_keep_alive nopoll
   ```

Establishing the ping interval for the object dispatcher

The default interval between object dispatcher pings is 180 seconds. If you have a slow network or a large number of managed nodes in a single Tivoli region that are frequently unavailable, you might increase the interval to 1800 seconds. (It is unlikely that you would ever decrease the interval.)

Note: Before changing this value, consult someone familiar with the Tivoli communications architecture, or contact your support provider.

Follow these steps to set the ping interval:

1. Log in to a managed node or the Tivoli server on which your Tivoli administrator has an alias with the **super** or **senior** role for the Tivoli region.
2. On UNIX operating systems only, ensure that your system environment is set correctly. For more information, see "Setting Tivoli environment variables" on page 1.
3. Enter the following command where *time* is the interval in seconds between managed node pings:

   ```
   odadmin set_keep_alive time
   ```

Considerations for communicating through firewalls

For security reasons, firewalls can terminate network connections between systems without notifying either system. When this happens, a process might continue to attempt to send data to the receiving process. However, the firewall discards this data without notifying the sending process. This ties up the resources of the sending process.

To avoid this problem, use the **odadmin** command and the **set_comm_check**, **set_comm_check_timeout**, and **set_comm_check_response_timeout** options. These options enable and configure a check that determines whether an inter-process communication (IPC) channel between two Tivoli processes is still active. Unlike the **odadmin set_keep_alive** option, these options check communication channels between all Tivoli processes, not just the communication between object dispatchers.

For a detailed description of the **odadmin** command and options, see the *Tivoli Management Framework Reference Manual*.

To properly configure the **odadmin** command and **set_comm_check**, **set_comm_check_timeout**, and **set_comm_check_response_timeout** options, and to troubleshoot communication problems, it is helpful to understand how these options work together. The following example describes the series of events that occur during the communication check. For this example, assume that the **set_comm_check_timeout** and **set_comm_check_response_timeout** options are both set to 60 seconds.

1. Tivoli Management Framework sets up the IPC channel.
2. The sending process sends a data packet.
3. After 60 seconds of inactivity on the IPC channel, the sending process pings the receiving process. The sending process sends this ping for the following reasons:
   - the **set_comm_check_timeout** of 60 seconds has elapsed, and
   - the sending process is ready to send another data packet. (The sending process sends a ping only if it is also sending data.)

After sending the ping, the sending process then sends the data without waiting for a response from the receiving process.

4. After 30 seconds of inactivity on the IPC channel, the sending process sends more data. The receiving process has not sent a response. However, the sending process sends data until the **set_comm_check_response_timeout** value is reached (in this case, 30 more seconds).
5. After 30 more seconds have elapsed and the **set_comm_check_response_timeout** value is reached, the sending process takes one of the following actions, depending on whether the receiving process has responded:
   - If the receiving process has not responded, the sending process determines that the IPC connection has been terminated. It frees the resources that it used to send data to the receiving process.
   - If the receiving process has responded, the sending process determines that the IPC connection is still valid and continues to send data.

Changing IP names and addresses

Changing machine IP names and addresses must be done carefully and in the appropriate manner. For example, if your Tivoli environment consists of hundreds of managed nodes in several regions, changing the name or subnet on which one of the Tivoli servers resides can have a significant impact on the installation.

You can change IP names and addresses in the following ways:

- Within the Tivoli environment using the Tivoli desktop (see the *Tivoli Management Framework User's Guide*)
- As the result of a Windows operating system running Dynamic Host Configuration Protocol (see the *Tivoli Management Framework Planning for Deployment Guide*)
- Outside the Tivoli environment using standard system commands (as described in the following sections)

If you have a choice, you should make IP address changes or system name changes through Tivoli software. The Tivoli environment automatically updates the managed node and Tivoli server.

However, sometimes changes occur outside the scope of your control. The following sections describe how to manually update the managed nodes and Tivoli server for changes made outside the Tivoli environment.

The following table provides the context and authorization role required for these tasks:

Activity	Context	Required Role
Change an IP address or alias of a managed node or Tivoli server from the desktop	Managed node	**super** or **senior**
Change an IP address or alias of a maaged node or Tivoli server using the **odadmin** command	Tivoli region	**super** or **senior**

You can change or add an IP alias from the command line only.

Changing the IP address of a managed node

If tools or processes outside of the Tivoli environment have changed the IP address of a managed node, use the **odadmin** command to inform the Tivoli server of the change. After you change the IP address and restart the machine, the object dispatcher service might not restart.

To correct this situation, shut down the object dispatcher on the managed node, run the **odadmin odlist** command on the Tivoli server, and then restart the object dispatcher on the managed node. For example, suppose that the IP address for the host named elvira was changed from 10.20.30.40 to 10.22.33.44 and that elvira is a managed node with dispatcher number 3. To inform the Tivoli server of the IP address change, follow these steps:

Note: For instructions on how to determine Tivoli dispatcher numbers, see "Listing active managed nodes" on page 7.

1. Log in to the Tivoli server as a user for which your Tivoli administrator has the **super** or **senior** role.
2. Ensure that your system environment is set correctly. For more information, see "Setting Tivoli environment variables" on page 1.
3. To change the recorded IP address for the managed node elvira, enter the following command:

   ```
   odadmin odlist change_ip 3 10.22.33.44
   ```
4. To start the object dispatcher on the managed node elvira, enter the following command:

   ```
   odadmin start 3
   ```

   Note: You cannot use the **odadmin** command to start the object dispatcher on a NetWare system. For instructions, see "Stopping and starting the object dispatcher" on page 8.

The Tivoli server for the region where the managed node elvira resides can now communicate with the object dispatcher on elvira to initiate management operations. For more information, see the **odadmin** command in the *Tivoli Management Framework Reference Manual*.

Changing the IP Address of a Tivoli server

To change the IP address of a Tivoli server, first notify the object dispatcher service of the change and then change the IP address of the Tivoli server. For example, suppose the IP address of the Tivoli server zeus is to be changed from 10.11.22.33 to 10.22.33.44. Follow these steps to notify the Tivoli server of the change, update the IP address, and restart the Tivoli services:

1. Log in to the Tivoli server as a user for which your Tivoli administrator has the **super** or **senior** role.
2. Ensure that your system environment is set correctly. For more information, see "Setting Tivoli environment variables" on page 1.
3. To shut down all object dispatcher services in the installation, enter the following command:

   ```
   odadmin shutdown clients
   ```
4. To notify the object dispatcher service on the server of the change, enter the following command, where 10.22.33.44 is the new IP address that you are going to change the Tivoli server to:

   ```
   odadmin odlist change_ip 1 10.22.33.44 FALSE
   ```

The **FALSE** option specifies that a **gethostbyaddr** function should not be performed on the new IP address yet, because the IP address has not actually been changed.

5. To change the IP address of the Tivoli server, edit the following files and restart the system:
   - Local /etc/hosts file
   - Network Information Services (NIS) hosts map
   - Domain Name Server (DNS) hosts map

   For Windows NT only, you also need to change the following file:
   - Local %SystemRoot%\system32\drivers\etc\lmhosts file
6. If necessary, enter the following command to restart the object dispatcher on the Tivoli server:
   ```
   odadmin start 1
   ```
   You also can start the object dispatcher on a specific operating system. For example:
   - On a UNIX Tivoli server, enter the following command:
     ```
     oserv.rc start
     ```
   - On a Windows Tivoli server, enter the following command:
     ```
     net start oserv
     ```
7. To restart the object dispatcher service on all the managed nodes, enter the following command:
   ```
   odadmin start clients
   ```

A worst-case scenario is one in which the IP address of the Tivoli server has already been changed. In this case, the object dispatcher process does not start. Follow these steps to synchronize the Tivoli server and managed nodes:

1. Log in to the Tivoli server as a user for which your Tivoli administrator has the **super** or **senior** role.
2. Ensure that your system environment is set correctly. See “Setting Tivoli environment variables” on page 1 for details.
3. To restart the object dispatcher on the Tivoli server, run one of the following commands. These commands restart the object dispatcher service on the Tivoli server and force the object dispatcher service to resolve both the server and all managed node IP addresses by calling the **gethostbyname** function with the name defined in the **odlist** entry.
   - On a UNIX Tivoli server, enter the following command:
     ```
     oserv -k $DBDIR -N by_name
     ```
   - On a Windows Tivoli server, enter the following command:
     ```
     net start oserv /-Nby_name
     ```
     Note: On Windows operating systems, do not add blank spaces after the forward slash (/) symbol.

   Notes:
   a. The shell variable **DBDIR** is set up by the Tivoli environment shell script, as described in “Setting Tivoli environment variables” on page 1.
   b. On Windows only, you do not need to specify the location of **DBDIR**. The object dispatcher reads the location from the registry.
4. To start the object dispatcher on all the managed nodes in the Tivoli region, enter the following command:
   ```
   odadmin start clients
   ```

Note: To start the object dispatcher on each managed node separately, enter the following command:

```
oserv -h host_name
```

where *host_name* specifies the name of the Tivoli server of the new Tivoli installation.

For more information, see the **odadmin** command in the *Tivoli Management Framework Reference Manual*.

Changing the IP addresses of all managed nodes in a region

If necessary, you can inform the Tivoli server and all the managed nodes within a Tivoli region of a change to their IP addresses. For example, you might want to do this if your organization restructures the subnets within your network.

To inform the Tivoli server and all the managed nodes of changes to all the IP addresses in the local Tivoli region, follow these steps:

1. Before changing any addresses, log in to the Tivoli server as a user for which your Tivoli administrator has the **super** or **senior** role.

   Note: If you have already changed the IP addresses, you must stop all the object dispatcher processes on the managed nodes and the Tivoli server manually. To shut down the object dispatcher, enter one of the following commands:
   - On a UNIX operating system, enter one of the following commands:
     ```
     /etc/Tivoli.oserv.rc stop
     ```
     ```
     odadmin shutdown
     ```
   - On a Windows NT system, enter the following command:
     ```
     net stop oserv
     ```

   Skip to step 5 when this is complete.
2. Ensure that your system environment is set correctly. See "Setting Tivoli environment variables" on page 1 for details.
3. To shut down the object dispatcher on the server and all managed nodes, enter the following command:
   ```
   odadmin shutdown all
   ```
4. Proceed with the network reorganization, changing IP addresses and subnets as appropriate.
5. To restart the Tivoli server, do one of the following:
   - On a UNIX Tivoli server, enter the following command:
     ```
     oserv -k $DBDIR -N by_name
     ```
   - On a Windows Tivoli server, enter the following command:
     ```
     net start oserv /-Nby_name
     ```

   This command restarts the object dispatcher service on the Tivoli server but forces it to resolve all the IP addresses by calling the **gethostbyname** function on the first or primary host name for each machine.

   Notes:
   a. The Tivoli environment shell script sets up the shell variable **DBDIR**, as described in "Setting Tivoli environment variables" on page 1.
   b. For Windows only, you do not need to specify the location of **DBDIR** in this command. The object dispatcher reads the location from the registry.

6. To restart the object dispatcher services on all managed nodes in the Tivoli region, enter the following command:

   ```
   odadmin start clients
   ```

For more information, see the **odadmin** command in the *Tivoli Management Framework Reference Manual*.

Changing the IP address of a connected Tivoli server

If you change the IP address of a Tivoli server that is connected to other regions, you must update the IP address of the Tivoli server on all connected regions. For example, suppose that you changed the IP address of a Tivoli server called Marketing that is connected to other regions.

To update the IP address of the Tivoli server on all connected regions, follow these steps:

1. Log in to the Tivoli server for the Marketing region as a user for which your Tivoli administrator has the **super** or **senior** role.
2. Ensure that your system environment is set correctly. See "Setting Tivoli environment variables" on page 1 for details.
3. Determine the Marketing region number, as described in "Viewing data for a Tivoli region" on page 4.
4. For each region connected to the Marketing region, log in to the Tivoli server for that region as a user for which your Tivoli administrator has set up the **super** or **senior** role.
5. To update the IP address of the changed region, enter the following command, where *region_num* is the region number of the Marketing region, *host* is the name of the Marketing Tivoli server, and 94 is the port number over which interregion communication is performed:

   ```
   odadmin region change_region region_num host 94
   ```

   Attention:: If you do not provide a region password when changing the IP address of an interconnected region, the **odadmin region change_region** command corrupts the odlist.dat file for the changed region.

In each Tivoli region in which this operation is performed, the Tivoli server performs the **gethostbyname** function, or if *host* is specified as an IP address, a **gethostbyaddr** function for the **Marketing** Tivoli server.

For more information, see the **odadmin** command in the *Tivoli Management Framework Reference Manual*.

Adding or removing IP aliases for a client or server

Sometimes you might choose to add an Ethernet card to a Tivoli server or managed node and define an IP alias to improve network performance. If you add such an IP alias and want the Tivoli server to communicate with the managed node using this address, you must inform the Tivoli server of the new address. If you do not inform the Tivoli server of the new IP alias and the managed node attempts to use the new Ethernet card to send a request, the Tivoli server rejects the request as unauthorized.

Note: If you want the Tivoli server to bind to a particular address, see the description of the `Force socket bind to a single address` statement in the

odadmin command output in "Viewing data for a Tivoli region" on page 4. For more information, see the **odadmin** command in the *Tivoli Management Framework Reference Manual*.

Suppose you add an Ethernet card and IP alias called elvira2 with address **10.22.33.44** for managed node elvira. Follow these steps to inform the Tivoli server of the change:

1. Log in to the Tivoli server for the Tivoli region in which elvira is located as a Tivoli administrator with the **super** or **senior** role.
2. Ensure that your system environment is set correctly. See "Setting Tivoli environment variables" on page 1 for details.
3. To list the currently known IP addresses and names, enter the following command:

   ```
   odadmin odlist
   ```

   Output similar to the following is displayed:

   ```
   1 Region   Disp  Flags  Port  IPaddr      Hostname(s)
   2 2000000000 1    ct-    94  10.84.44.18   yogi.tivoli.com
   3            2    ct-    94  10.84.44.30   bear.tivoli.com
   4            3    ct-    94  10.84.33.38   elvira.tivoli.com
   5            4    ct-    94  10.84.33.22   fuji.tivoli.com
   ```
4. To add the IP alias for elvira2, enter the following command, where *alias* is either the alias name (elvira2) or the IP address (10.22.33.44):

   ```
   odadmin odlist add_ip_alias 3 alias
   ```
5. To list the known IP addresses and names, enter the **odadmin** command again:

   ```
   odadmin odlist
   ```

   Output similar to the following is displayed:

   ```
   1 Region   Disp  Flags  Port  IPaddr       Hostname(s)
   2 2000000000 1    ct-  94    10.84.44.18   yogi.tivoli.com
   3            2    ct-  94    10.84.44.30   bear.tivoli.com
   4            3    ct-  94    10.84.33.38   elvira.tivoli.com
   5                            10.22.33.44   elvira2.tivoli.com,elvira2
   6            4    ct-  94    10.84.33.22   fuji.tivoli.com
   ```

For each managed node that you add an IP alias, the Tivoli server performs a **gethostbyname** function and stores the new resolved IP address.

Suppose you later have to remove the Ethernet card and IP alias elvira2 for the managed node elvira. Follow these steps to inform the Tivoli server of the change:

1. Log in to the Tivoli server for the Tivoli region in which elvira is located as a user for which your Tivoli administrator has the **super** or **senior** role.
2. Ensure that your system environment is set correctly. See "Setting Tivoli environment variables" on page 1 for details.
3. To remove the IP alias for elvira2, enter the following command where *od_num* is the object dispatcher number:

   ```
   odadmin odlist delete_ip_alias od_num 10.22.33.44
   ```

When you remove an IP alias, the Tivoli server removes the IP address from its internal table. You can run the **odadmin odlist** command as shown to list the updated IP address and name table maintained by the Tivoli server.

For more information, see the **odadmin** command in the *Tivoli Management Framework Reference Manual*.

Changing the IP Alias of a Connected Tivoli Server

If you add the IP alias of a Tivoli server that is connected to other Tivoli regions, you must tell each connected Tivoli region the new IP alias of the local Tivoli server.

Suppose you have changed the IP alias of the Tivoli server in a region called **Marketing**. Follow these steps to update the Tivoli server IP address on all connected Tivoli regions:

1. Log in to the Tivoli server for the **Marketing** Tivoli region as a user for which your Tivoli administrator has an alias with the **super** or **senior** role.
2. Ensure that your system environment is set correctly. See "Setting Tivoli environment variables" on page 1 for details.
3. Determine the **Marketing** region number, as described in "Viewing data for a Tivoli region" on page 4.
4. For each Tivoli region connected to the **Marketing** region, log in to the Tivoli server for that region as a user for which your Tivoli administrator has the **super** or **senior** role.
5. To update the changed region IP alias, enter the following command, where *region* is the region of the **Marketing** region, *IPaddr* is the address of the **Marketing** Tivoli server, and *hostname* is the name of the **Marketing** Tivoli server.

   ```
   odadmin region add_IP_alias region IPaddr hostname
   ```

For more information, see the **odadmin** command in the *Tivoli Management Framework Reference Manual*.

Adding or Removing Host Name Aliases for a Client or Server

You might have to add a new host name alias to a managed node or Tivoli server. If you add such a host name alias and want the Tivoli server to communicate with the managed node using this name, you must inform the Tivoli server of the new alias.

Suppose you add the alias elvira3 for the managed node elvira. To inform the Tivoli server, follow these steps:

1. Log in to the Tivoli server for the region in which elvira is a user with an alias and on which your Tivoli administrator has an alias with the **super** or **senior** role.
2. Ensure that your system environment is set correctly. See "Setting Tivoli environment variables" on page 1 for details.
3. To list the currently known IP addresses and names, enter the following command:

   ```
   odadmin odlist
   ```

   Output similar to the following is displayed:

   ```
   1 Region   Disp Flags Port  IPaddr       Hostname(s)
   2 2000000000 1   ct-  94    10.84.44.18   yogi.tivoli.com
   3            2   ct-  94    10.84.44.30   bear.tivoli.com
   4            3   ct-  94    10.84.33.38   elvira.tivoli.com
   5            4   ct-  94    10.84.33.22   fuji.tivoli.com
   ```
4. To add the host name alias elvira3, enter the following command:

   ```
   odadmin odlist add_hostname_alias 3 10.84.33.38 elvira3
   ```
5. To list the known IP addresses and names, enter the **odadmin** command again:

   ```
   odadmin odlist
   ```

Output similar to the following is displayed:

```
1 Region     Disp Flags Port  IPaddr       Hostname(s)
2 2000000000 1    ct-   94    10.84.44.18  yogi.tivoli.com
3            2    ct-   94    10.84.44.30  bear.tivoli.com
4            3    ct-   94    10.84.33.38  elvira.tivoli.com,elvira3
5            4    ct-   94    10.84.33.22  fuji.tivoli.com
```

Suppose you later have to remove the alias elvira3 for the managed node elvira. Follow these steps to inform the Tivoli server:

1. Log in to the Tivoli server for the region in which elvira is located as a user for which your Tivoli administrator has an alias with the **super** or **senior** role.
2. Ensure that your system environment is set correctly. See "Setting Tivoli environment variables" on page 1 for details.
3. To remove the host name alias for elvira3, enter the following command:
   ```
   odadmin odlist delete_hostname_alias 3 10.84.33.38 elvira3
   ```

When you remove a host name alias, the Tivoli server removes the alias from its internal table. It is recommended that you test these procedures in a test Tivoli region to increase the ease and success of these types of complicated changes.

For more information, see the **odadmin** command in the *Tivoli Management Framework Reference Manual*.

Changing the Tivoli server configuration

Due to machine upgrades, changing management requirements, workloads, and file system reorganizations, it might become necessary to change the file systems in which the Tivoli binaries, libraries, and database on the Tivoli server are located or to move the Tivoli server for a region from one machine to another.

If you decide to change anything about the Tivoli server configuration or the machine that functions as the Tivoli server, consider the following points:

- Move the Tivoli server to a machine of the same architecture type as the original.
- Move the Tivoli server only after careful analysis. Make sure that you plan and double-check your steps when performing this procedure.
- When you move the Tivoli binaries, libraries, or database to a different file system, you must change some of the configuration information stored in the Tivoli database.

The following table provides the context and authorization roles required for these tasks.

Activity	Context	Required Role
Upgrade the operating system of a Tivoli server or managed node	Tivoli region	**root** access or **Administrator** privileges
Move the Tivoli database, binaries, libraries, and other installation directories on the Tivoli server or managed node to a new file system	Tivoli region	**root** access or **administrator** privileges
Change the name of a managed node	Policy region	admin, **super** or **senior** in the policy region in which the managed node is located

Changing the operating system of a Tivoli server

When upgrading an operating system from one version to another, it is important that you review the Tivoli Management Framework release notes to ensure that your system is at the appropriate level to support Tivoli Management Framework. These notes provide information about which operating systems are supported and which interpreter types map to the various operating system versions.

If you plan to upgrade from a sunos4 interpreter type to solaris2, the interpreter type changes and you must follow the upgrade instructions outlined in this section. If you plan to upgrade between similar operating systems, you do not need to perform these prerequisite steps.

Before upgrading operating systems, you should make a backup of your system and any important Tivoli customizations, including the following:

Tasks
Tasks are stored in *install_dir*/$INTERP/TAS/TASKLIBRARY/. Upgrading the operating system changes the **$INTERP** for the machine. Tasks defined to the old interpreter type are in the old $BINDIR path. As long as this path is not removed, these tasks are valid. When the tasks are binary, they might not work on the new interpreter type. If this is the case, you must create a new task or new binary on the new version of the operating system.

Rule Bases and Adapter Configuration Files
You must change the definitions of rule bases in $BINDIR after upgrading the operating system. The paths to rule bases are stored in the Tivoli database, and the new default rule base is defined with the new $BINDIR in the path. After the old database is restored, interp_upgrade.sh (located in the $BINDIR/TAS/INSTALL directory) takes care of the default rule base path, but you must move or update the paths to custom rule bases using the **wsetrb** command.

RIM Information
When upgrading an operating system, it might be necessary to upgrade other applications. This is important with respect to relational databases, such as Sybase and Oracle. For example, if a machine is to be upgraded from AIX 3.2.5 to AIX 4.*x*, you also must install a new version of Sybase to run on AIX 4.*x*. If the Sybase database is installed to a different location, you must update the RIM object. This also applies if Sybase is reinstalled at the same version level, but to a different location.

Generally, if the location of the database for your product changes, you might have to update the Tivoli environment as well using the RIM object and the **wsetrim** command.

Custom Monitor Collections
As with any customizations made to the Tivoli environment, customizations to monitors or custom collections should be preserved. If these definition files are located in $BINDIR, you might want to move them to the new $BINDIR.

Tivoli Managers
If Tivoli managers are installed, ensure that they are available for the new operating system before you upgrade. You should also ensure that the application the Tivoli manager manages is available for the new operating system.

Note: If there are any Tivoli customizations, such as method bodies stored in $BINDIR, move these to the new $BINDIR directory.

Upgrading your operating system

To upgrade your operating system, follow these steps:

1. Ensure that your system environment is set correctly. See "Setting Tivoli environment variables" on page 1 for details.
2. To list the products and patches installed in a Tivoli region, enter the following command:
   ```
   wlsinst -ah
   ```
   Note: Redirect this output to a file or print this output. This information is required for the managed node or Tivoli server being updated.
3. To shut down the Tivoli server, enter one of the following commands:
   - `odadmin shutdown 1`
   - `/etc/Tivoli/oserv.rc stop`
   - `net stop oserv`
4. To back up the current database directory ($BINDIR), enter the following commands:
   - `cd $DBDIR/..`
   - `tar -cvf` *hostname*`.db.tar` *hostname*`.db`
5. To test untarring this image, enter the following command:
   `tar -tvf` *hostname*`.db.tar`

   After testing this image, store it in a safe location.
6. Install a new Tivoli server on the new version of the operating system. For installation instructions, see the *Tivoli Enterprise Installation Guide*.
7. Install all the patches and products that were previously installed.
8. Shut down the Tivoli server as described in step 3.
9. To replace the original Tivoli database, follow these steps:
   a. Change the name of the current database directory.
   b. Untar the old database.
   c. For Windows operating systems only, copy the msvcirt.dll, msvcrt.dll, msvcrt40.dll, libuthreads.dll, libuthreads60.dll, and oserv.exe files from the new database directory to the old database directory.
   d. Enter the following upgrade script:
   ```
   $BINDIR/TAS/INSTALL/interp_upgrade.sh
   ```

Moving Tivoli directories

Occasionally, you might need to move either the binaries, libraries, or database of a Tivoli server or managed node from one directory to another or to another machine. In addition, you also might need to move other installation locations, such as directories for message catalogs, application defaults, and manual pages.

You can move binaries, libraries, and databases from one directory to another. You also can move the binaries and libraries off of the Tivoli server or managed node onto another machine. You must, however, make sure that the Tivoli server or managed node can access the files in the new location by ensuring that the new directories are properly mounted.

In addition, you can move the database to another machine if you are planning to move the Tivoli server or managed node to that machine.

This section provides instructions about how to move these files. You can combine these procedures if you plan to move these files and directories at the same time.

The following is the output of the **odadmin** command from a UNIX operating system. The entries shown in bold are the entries that are changed during this process.

```
Region = 1158603981
Dispatcher = 1
Interpreter type = solaris2
Database directory = /var/spool/Tivoli/emi.db
Install directory = /us/local/Tivoli/bin
Inter-dispatcher encryption level = simple
Kerberos in use = FALSE
Remote client login allowed = TRUE
Install library path =/usr/local/Tivoli/lib/solaris2b
Force socket bind to a single address = FALSE
Perform local hostname lookup for IOM connections = FALSE
Tivoli Management Framework () #1 Wed Mar 1 15:20:57  2000
Copyright Tivoli Systems, 1999.
Port range = (not restricted)
State flags in use = TRUE
State checking in use = TRUE
State checking every 180 seconds
Dynamic IP addressing allowed = FALSE
Transaction manager will retry messages 4 times.
```

Note: It is recommended that you install the database locally on the Tivoli server or managed node. Do not move this directory unless you are planning to move the Tivoli server or managed node to another machine.

Moving the Tivoli database

To move the database of a Tivoli server or managed node, follow these steps:

1. Make a backup of the Tivoli server or managed node, which is being moved. For instructions, see "Backing up Tivoli databases" on page 35.
2. Log in as **root** or **administrator** on the Tivoli server or managed node.
3. Ensure that your system environment is set correctly. See "Setting Tivoli environment variables" on page 1 for details.
4. Do one of the following:
   - If you are moving the database of a managed node, shut the managed node down by entering one of the following commands:
     – On a UNIX operating system, enter the following command:
       ```
       /etc/Tivoli/oserv.rc stop
       ```
     – On a Windows operating system, enter the following command:
       ```
       net stop oserv
       ```
   - If you are moving the database of a Tivoli server, shut down the Tivoli server and all its managed nodes by entering the following command:
     ```
     odadmin shutdown all
     ```
5. Copy or move the directory files to the new location or file system.
6. Edit the following files to adjust the paths for the directory:

   UNIX (except HP-UX)
   /etc/Tivoli/setup_env.sh, /etc/Tivoli/setup_env.csh, /etc/Tivoli/oserv.rc

HP-UX 10
: /sbin/init.d/Tivoli

Windows NT, Windows XP, and Windows 2000
: $SystemRoot/system32/drivers/etc/Tivoli/ setup_env.cmd, $SystemRoot/system32/ drivers/etc/Tivoli/setup_env.sh

In particular, make sure that the paths specified for the environment variables **DBDIR**, **BINDIR**, and **LIBDIR** are correct.

7. To reestablish the correct values for the environment variables in the Tivoli environment, repeat step 3.
8. To start the object dispatcher, do one of the following:
   - On a UNIX operating system, enter the following command:
     ```
     oserv -k $DBDIR
     ```
   - On a Windows operating system, follow these steps:
     a. Select **Start → Run**, type **regedit**, and click **OK**. The Registry Editor window is displayed.
     b. Select these folders (in the following order) to display the registry values for the database and service directories:
        ```
        HKEY_LOCAL_MACHINE→SOFTWARE→TIVOLI→PLATFORM→OSERV94
        ```
     c. Double-click **Database directory** and change the value data to point to the new location. Repeat this step to change the Service directory path.
     d. Select these folders (in the following order) to display the registry values for the **ImagePath** value data:
        ```
        HKEY_LOCAL_MACHINE→SYSTEM→CurrentControlSet→Services→oserv
        ```
     e. Double-click **ImagePath** and change the value data to point to the new location.
     f. To start the object dispatcher, enter the following command:
        ```
        net start oserv
        ```
9. To update the location stored in the database that is used to support versioned system files when profiles are distributed, follow these steps:
   a. To change to the database directory, enter the following command:
      ```
      cd $DBDIR
      ```
   b. To return the object ID for the **fileioRef** object, enter the following command:
      ```
      VS=`idlattr -t -g $TMR.host#.0 fileioRef Object`
      ```
   c. To view the current path, enter the following command:
      ```
      idlattr -t -g $VS versioning_area string
      ```
   d. To set the new version directory path, enter the following command, where *newpath* is the absolute path name as represented by the **DBDIR** environment variable:
      ```
      idlattr -t -s $VS versioning_area string "newpath/file_versions"
      ```
10. If you are moving the directory of a Tivoli server only, change the default backup directory with the following commands:
    a. To retrieve the backup class object identifier, enter the following command:
       ```
       backup_OID=`wlookup TMRBackup`
       ```
    b. To return the current backup path, enter the following command:
       ```
       idlattr -tg $backup_OID default_device string
       ```
    c. To set the new backup directory path, enter the following command:
       ```
       idlattr -ts $backup_OID default_device string '"newpath"'
       ```

11. Restart the managed nodes in the Tivoli region. For instructions, see "Stopping and starting the object dispatcher" on page 8.
12. Download and run the update_locations.sh script. The **update_locations.sh** script is available on IBM anonymous FTP server.

    If you are unfamiliar with anonymous FTP or do not have Internet access, contact your support provider for a copy of this script.

Moving Tivoli binaries

To move Tivoli binaries on a Tivoli server or managed node, follow these steps:

1. Follow steps 1 through 7 in "Moving the Tivoli database" on page 26.
2. To start the object dispatcher, do one of the following:
   - On a UNIX Tivoli server, enter the following command where *install_directory* is the path listed in the **odadmin** output on the managed node.:
     ```
     oserv -k $DBDIR -b install_directory
     ```
   - On a Windows Tivoli server, enter the following command where *install_directory* is the path listed in the **odadmin** output on the managed node.:
     ```
     net start oserv /-b install_directory
     ```
3. Restart the managed nodes in the Tivoli region. For instructions, see "Stopping and starting the object dispatcher" on page 8.
4. Download and run the **update_locations.sh** script. The **update_locations.sh** script is available on the IBM anonymous FTP server.

   If you are unfamiliar with anonymous FTP or do not have Internet access, contact your support provider for a copy of this tool.

Moving Tivoli libraries on UNIX systems

To move Tivoli libraries on a UNIX Tivoli server or managed node, follow these steps:

Note: Windows operating system do not contain libraries. Dynamically linked libraries (DLLs) are stored in the binary directory instead of a separate library directory.

1. Follow steps 1 through 7 in "Moving the Tivoli database" on page 26.
2. To start the object dispatcher, enter one of the following commands (dependent upon the operating system running on the Tivoli server):

   AIX
   ```
   oserv -k $DBDIR -B $LIBPATH
   ```
   HP-UX
   ```
   oserv -k $DBDIR -B $SHLIB_PATH
   ```
   SunOS
   ```
   oserv -k $DBDIR -B $LD_LIBRARY_PATH
   ```
   SVR4
   ```
   oserv -k $DBDIR -B $LD_LIBRARY_PATH
   ```
3. Restart the managed nodes in the Tivoli region. For instructions, see "Stopping and starting the object dispatcher" on page 8.
4. Download and run the **update_locations.sh** script. The **update_locations.sh** script is available on the IBM support FTP server.

   If you are unfamiliar with anonymous FTP or do not have Internet access, contact your support provider for a copy of this script.

Moving other installation directories

To move message catalogs, application defaults, manual pages, and any other installation directories, follow these steps:

1. Follow steps 1 through 7 in "Moving the Tivoli database" on page 26.
2. Restart the managed nodes in the Tivoli region. For instructions, see "Stopping and starting the object dispatcher" on page 8.
3. Download and run the **update_locations.sh** script. The **update_locations.sh** script is available on the IBM anonymous FTP server.

   If you are unfamiliar with anonymous FTP or do not have Internet access, contact your support provider for a copy of this script.

You have now moved installation directories to a new location. It is recommended that you make a backup of the Tivoli region using the procedure outlined in "Backing up Tivoli databases" on page 35 before proceeding. Performing the backup now ensures that you do not have to repeat these steps should something corrupt your environment while moving the Tivoli server.

Changing the name of a managed node or endpoint

You can change the name of a managed node or endpoint. However, changing the name of your Tivoli server is not recommended. For more information, contact your support provider.

To change the name of a managed node, follow these steps:

1. To change the label on a managed node, enter the following command:

   ```
   MNoid=`wlookup -r ManagedNode old-name`
   idlcall $MNoid _set_label '"new-name"
   ```
2. To ensure that the local host variables reflect the new name of the managed node, do one of the following:
   - On UNIX operating systems, ensure that the IP address matches the WLOCALHOST variable.
   - On Windows operating systems, enter the following command to set the local host name in the Windows registry, where *host_name* is the name of the local host or its IP address:

     ```
     wlocalhost host_name
     ```

     If you do not specify *host_name*, **wlocalhost** returns the host name currently stored in the registry. For more information about host name resolution, see the *Tivoli Management Framework Planning for Deployment Guide*.
3. Change the alias of the managed node by adding a new alias and deleting the old one. For complete instructions, see "Adding or Removing Host Name Aliases for a Client or Server" on page 22.

The name of the managed node is now changed. To change the name of the database to match the new managed node name, see "Moving Tivoli directories" on page 25.

To change the name of an endpoint, follow these steps:

1. To change the label on a endpoint, enter the following command, where *new_label* changes the current endpoint label (*ep_label*) to the new label specified by *new_label*:

   ```
   wep ep_label set_label new_label
   ```
2. To synchronize the endpoint data stored by the endpoint manager, gateways, and endpoints within a Tivoli region, enter the following command:

```
wep sync_gateways
```

3. Change the host name alias by adding the new alias and deleting the old alias. For complete instructions, see "Adding or Removing Host Name Aliases for a Client or Server" on page 22.

The name of the endpoint is now changed. To change the name of the database to match the new managed node name, see "Moving Tivoli directories" on page 25.

Deleting versioned system files

Tivoli Management Framework provides file input/output routines for tracking and controlling revisions to system files created by a Tivoli application. The change history for each system file is maintained in the Revision Control System (RCS). Each time, for example, that the user administration application updates the UNIX /etc/passwd file on a system, the previous version is maintained under RCS control. Use the RCS tools provided with Tivoli Management Framework to recover previous revisions of a system file if problems occur.

The versioning of system files by Tivoli applications is completely automatic. The updated system file is placed in the standard location on the system with the appropriate structure and contents. The versioned copy of the system file is maintained under the Tivoli database directory in a special directory called /file_versions. For example, if your database is stored in the $DBDIR directory, the versioned file for the /etc/passwd system file is located in $DBDIR/file_versions/etc/RCS/passwd,v. To determine the database directory, enter the **odadmin odinfo** command, as described in "Viewing data for a Tivoli region" on page 4.

Occasionally search the RCS log for the versioned system files generated by one or more Tivoli applications. After several months the versioned password file might contain several hundred deltas reflecting the changes that have been made over the elapsed period of time. Refer to your Tivoli applications manuals for information about which system files are modified and how they are handled.

You can remove old versions of modified system files from the command line only.

To delete old versions of a modified system file, follow these steps:

1. Log in as root or administrator to a managed node or the Tivoli server on which you want to delete a versioned system file.
2. Ensure that your system environment is set correctly. For more information, see "Setting Tivoli environment variables" on page 1.
3. Change directories to the local Tivoli database directory. Use the **odadmin odinfo** command, as described in "Viewing data for a Tivoli region" on page 4, to find the database directory.
4. Change directories to the versioned files subdirectory:
   ```
   cd file_versions
   ```

   This subdirectory contains a number of other subdirectories, one for each system directory that contains system files modified by a Tivoli-based application. For example, the user administration application modifies the /etc/group file, so there is an /etc subdirectory in the /file_versions subdirectory. In this directory is an /RCS subdirectory where versioned system files from the /etc system directory are stored.
5. Change directories and go to the /etc/RCS subdirectory:
   ```
   cd etc/RCS
   ```

Each file in this directory has a ,v extension. This is the naming convention used by RCS to indicate a versioned file.

6. To review the versions currently maintained in the group,v file, enter the following command:

   ```
   wrlog group
   ```

   Output similar to the following is displayed:

   ```
   RCS file: group,v
   Working file: group
   head: 1.4
   branch:
   locks: strict
   root: 1.4
   access list:
   symbolic names:
   comment leader: "# "
   keyword substitution: kv
   total revisions: 4;     selected revisions: 4
   description:
   ----------------------------
   revision 1.4    locked by: root;
   date: 1994/09/02 22:52:48;  author: jim;  state:
   Exp;  lines: +2 -1
   Changed by jim@tivoli.
   ----------------------------
   revision 1.3
   date: 1994/09/02 22:52:29;  author: mary;  state:
   Exp;  lines: +1 -0
   Changed by mary@tivoli.
   ----------------------------
   revision 1.2
   date: 1994/09/02 22:16:14;  author: jim;  state:
   Exp;  lines: +13 -10
   Changed by jim@tivoli.
   ----------------------------
   revision 1.1
   date: 1994/09/02 22:12:12;  author: jim;  state:
   Exp;
   Changed by jim@tivoli.
   ```

   The main items of interest are the names of the Tivoli administrators who made modifications to this file. This example output contains four versions of the group file. The jim@tivoli administrator made three changes, and the mary@tivoli administrator made one change.

7. To remove the two oldest versions, 1.1 and 1.2, from the versioned file, enter the following command:

   ```
   % wrcs -o1.1:1.2 group
   RCS file: group,v
   deleting revision 1.2
   deleting revision 1.1
   done
   ```

8. To confirm the old revisions have been deleted, enter the following command:

   ```
   wrlog
   ```

   Output similar to the following is displayed:

   ```
   RCS file: group,v
   Working file: group
   head: 1.4
   branch:
   locks: strict
   root: 1.4
   access list:
   symbolic names:
   ```

```
comment leader: "# "
keyword substitution: kv
total revisions: 2;     selected revisions: 2
description:
----------------------------
revision 1.4    locked by: root;
date: 1994/09/02 22:52:48;  author: root;  state:
Exp;  lines: +2 -1
Changed by root@tivoli.
----------------------------
revision 1.3
date: 1994/09/02 22:52:29;  author: root;  state:
Exp;
Changed by root@tivoli.
```

For more information about the **odadmin**, **wrlog**, and **wrcs** commands, see *Tivoli Management Framework Reference Manual*.

Improving performance in a Tivoli environment

Because your Tivoli environment becomes more complex as you add more Tivoli applications and systems managed by Tivoli software, you should revisit your initial hardware configuration to improve performance. The following sections discuss options to consider.

Note: Contact your support providers to discuss how these changes might affect your Tivoli environment.

Process tables limits (UNIX only)

One of the ways you can improve how the Tivoli environment runs on UNIX systems is to monitor the process tables for the Tivoli user and the entire system. Check to make sure that your system-wide process table is large enough, and then check that the per-user process limit is also high enough.

The following guidelines provide information about how to analyze the process tables (see your system documentation on how to run the **sar**, **ulimit**, and **ps** commands or other process management commands particular to your system):

1. While the Tivoli environment is running, check the overall number of processes.
2. Establish how many processes are running on the system.
3. Determine how many processes are running as the Tivoli user (the **tmersrvd** account on HPUX and the **nobody** account on AIX and Solaris).
4. Compare how many processes are currently running to how many processes are allowed to run. Make this comparison for both the system-wide process table and per-user process limit. When the limits are exceeded, you might need to change the number of processes allowed to run.

File table limits (UNIX only)

Another area where you can optimize performance for UNIX systems is in the configuration for the file table. Not only should you check the number of files open on the file system, but you should also check every socket connection that is open. When performing this check, run the **netstat** command to see how many socket connections are open. Also run the **sar –v** command (or your system’s equivalent) to see file-handle-table statistics and process table statistics, among others. If your file table is approaching the limit, raise it and regenerate your kernel. Take into consideration your soft-file handle limit, if your operating system has both a hard and a soft limit. Set limits so that neither is approached.

Note: Tools that list the number of open files per process are available on the Internet.

Swap space used

Another performance problem can be found in the swap-space configuration. Check the amount of configured swap space, considering whether it is the primary or secondary device or file system. If you are using 90% to 100%, you need more swap space.

New processes, when spawned, need three types of memory allocated:
- Memory in physical RAM for code and data space
- Memory in swap space for code and data
- Memory in swap space reserved for code and data

Virtual memory and paging size (Windows only)

When you find that a Windows managed node or Tivoli server is running low on virtual memory or is hanging, you might need to increase the virtual memory on the machine. You also should consider increasing the total paging file size for the system.

Processing load for applications

When you have multiple Tivoli applications installed, you might have problems with the applications successfully completing their tasks, such as software distributions or inventory scans. It is important to properly schedule application tasks so that they do not process at the same time. This can greatly reduce resource problems in the Tivoli environment.

Physical RAM constraints

Tivoli Management Framework requires space to expand as your Tivoli environment becomes more complex. It needs space to spawn more processes (process and file tables, RAM, and swap space), create new threads (RAM, swap space, and file handles), and so on.

It is important to check for RAM usage. To improve performance in the Tivoli environment, try any of the following:
- Run a Tivoli desktop from a managed node instead of from the Tivoli server. Servicing desktops requires many threads and database activity.
- Eliminate unnecessary drivers from your kernel and eliminate unnecessary operating system processes. For example, if you do not intend to use a system as an NFS server, disable the NFS server subsystem.
- If you are booting or running Xterminal (xterm) from your Tivoli server, move it to a different system—xterm windows consume a large amount of RAM.

Performance considerations with gateways

Every time a gateway runs a method on an endpoint, the gateway performs a dependency check to see if the files required by the method already exist in the endpoint cache. To perform the dependency check, the gateway checks the version of each prerequisite file in the appropriate bundle and then checks the cache on the endpoint for the same file and version. If the prerequisite files already exist on the endpoint, the gateway runs the method on the endpoint. If one or more prerequisite files are missing from the endpoint, the gateway places the files on the endpoint before running the method.

The **wgateway** *gateway_name* **set depcache** command enables you to increase the efficiency of this dependency check. When this parameter is set to `TRUE`, the gateway creates a hash table that contains the name and version of the files in each bundle. Then, when the gateway runs an endpoint method, it reads information about the files from the hash table instead of from the bundles.

To view this hash table, run one of the following commands:

wgateway *gateway_name* **run depcache dump**
: This command dumps the contents of the hash table to a file named depcache_dump in the DBDIR directory.

wgateway *gateway_name* **run depcache dump_in_gatelog**
: This command dumps the contents of the hash table to the gatelog.

For more information about the **wgateway** command, see the *Tivoli Management Framework Reference Manual*.

Performance considerations with endpoints

When you encounter problems with endpoint logins, there could be a problem with the Tivoli server being loaded down with too many requests. You can use additional throttling options to better distribute the load of endpoint logins being handled by the endpoint manager at one time. Use the **wepmgr** command with the **set** option to define attribute values, **login_interval**, **max_install**, **max_sgp**, **max_after**, and **max_jobs**. Setting these attributes assist you in throttling endpoint login requests.

Another consideration is how to implement endpoint policy. It is recommended that you use Perl scripts if pattern match searches are required. Perl is a compiled language, as contrasted with shell scripts, which are interpreted. Perl is therefore faster. In addition, while each command used in a shell script requires spawning of an entire child process (a resource-intensive activity), a compiled Perl script runs as one process, which makes it system-friendly. Perl is also much more portable across gateways, regardless of the interpreter type.

Chapter 2. Maintaining the Tivoli database

A Tivoli region stores management data and resource descriptions in a distributed, persistent database. The main management database for the Tivoli environment resides on the Tivoli server, with local databases residing on each managed node. The object database on the Tivoli server contains all the objects from the entire Tivoli region. Each managed node keeps a subset of objects related to itself. Tivoli Management Framework uses a distributed transaction service to ensure that modifications to the distributed databases are consistent and complete. Thus, if a database operation fails on a local database, the main Tivoli database is not affected with a partial update.

Without adequate and regular attention to the Tivoli database, many problems can occur in the Tivoli environment. Part of any disaster recovery plan should recognize and account for maintaining the Tivoli database. If key hardware or software components fail in the Tivoli environment, having a regularly maintained and backed up database can help minimize costly downtime.

This chapter explains how to perform the following tasks:
- "Backing up Tivoli databases"
- "Restoring Databases in a Region" on page 41
- "Troubleshooting Backup and Restore Operations" on page 49
- "Repairing the Tivoli Database" on page 52

Note: For information about business recovery system planning, see the *Tivoli Management Framework Planning for Deployment Guide.*

Backing up Tivoli databases

The Tivoli region relies on the Tivoli database to successfully manage computer systems. Like all sensitive data, databases should be regularly backed up and archived in case the information becomes lost or damaged. If this happens, the Tivoli environment loses track of its resources and the applications it is using to manage the resources. Therefore, you should perform backups often while developing the Tivoli environment and regularly after deployment.

You can back up one managed node, several managed nodes, the Tivoli server, or the entire Tivoli region from either the Tivoli desktop or the command line. In addition, you can perform backups immediately or schedule a backup. Note that no data is stored on an endpoint. Therefore, the backup mechanism discussed in this section only applies to managed nodes in the local Tivoli region. You must back up these databases on a regular scheduled basis as well as before and immediately after significant changes, such as:
- Installation of products
- Major maintenance procedures
- Creation of profiles
- Creation of multiple managed nodes

Make daily backups of your Tivoli servers. Also, schedule backups for managed nodes on a regular basis, during quiet periods of operation. In a large Tivoli environment, you might choose to back up a different group of managed nodes each night. When all the managed nodes have been backed up, you can start over with the first group of managed nodes.

These backups should be kept on a separate tape and in a secure place for the life of the system. The backup could be the only way to rescue a management machine in some circumstances.

If you need to restore the information for a specific machine, perhaps because its operating system has been upgraded or a disk drive replaced, Tivoli Management Framework provides a restore operation through the command line. There are two methods for restoring a database. The first is the standard method to use if a system is otherwise operational (and the object dispatcher is running), which is described on "Restoring a database when the object dispatcher is running" on page 43. The second is referred to as a rescue operation and is used if the object dispatcher cannot be started. See "Restoring a database when the object dispatcher is down" on page 43 for details.

Notes:

1. A Tivoli backup backs up only information in the Tivoli database. It does not back up other Tivoli data such as binaries or libraries. You should include all Tivoli servers and file servers in the full system backups normally performed in your organization using a backup solution.
2. You cannot store backup images on a NetWare managed node. For example, if you select **Desktop → Backup** from the Tivoli desktop, you cannot specify the name of a NetWare managed node in the **Save image on node** field.

Tivoli Backup Process

Backups are performed on the Tivoli server and managed node database files. Because the Tivoli object dispatcher can be writing to the Tivoli database at any time, use the built-in backup mechanism instead of a simple tar or cpio file. This mechanism uses a snapshot process to capture the data while it is stable to prevent the loss of data.

Note: If you need to do a tar or cpio backup, first shut down the oserv service.

In a large Tivoli region, the **wbkupdb** command process can take a long time because backups of the managed node databases are done serially. Concurrent backup processes are not allowed within the same region. A **wbkupdb** command must be completed before another can be run.

During a backup operation, it creates a temporary backup file as a staging site while it copies the databases. If the destination directory does not have enough space, you can change the destination directory by setting the **TMPDIR** environment variable. See "Changing the Destination of a Temporary Backup File" on page 41 for additional instructions.

After the temporary backup file is written, the backup operation moves the archived backup file to the backups directory. The name of the backup file is **DB_***date-time*. The backups directory is created in the same directory that contains the database. For example, using the default installation path, the path and file name of the backup file might be /var/spool/Tivoli/backups/DB_Oct11-1311. If you perform the backup from the command line, you can specify the file name and destination directory by using the **wbkupdb** –d option.

Tivoli Management Framework does not automatically delete old backup files. You should periodically delete these outdated files. You might, for example, create a task that runs once a week and deletes any backup files that are more than 7 days older than the current date.

The following table provides the context and authorization roles required for this task:

Activity	Context	Required Role
Back up one or more managed nodes, the Tivoli server, or the entire Tivoli region	Tivoli region	**backup**, **super**

A number of security features are used in the backup process. These features are as follows:

- The administrator must have a valid user login name and a group name for the machine where the backup file are stored.

  To change the user login or group name, open the Administrators window, right-click the administrator icon and select **Edit Properties**. Be sure that you have the desired user login names, and then restart the Tivoli desktop if a change was made.
- On UNIX operating systems, use the **umask** command to set the root account to 022.

Considerations for Backing up Connected Regions

In a Tivoli environment with connected Tivoli regions, consider additional items before planning and running database backup and restore operations. Because of the complex nature of exchanging resources between regions, these resources are registered in each Tivoli region that participates in the resource exchange. When performing a backup in a multiple region environment, determine when scheduled or manual resource exchanges occur and schedule your backups to occur after these resource exchanges.

The following are some problems that can occur if you restore from a backup in only one Tivoli region:

- Subscriptions to a connected Tivoli region could become down-level after a restore operation.
- Profile relationships between connected Tivoli regions could become out of synchronization after a restore operation.
- New installations of Tivoli applications in one Tivoli region could become nonoperational after a restore operation, if the application depends on resources in a connected region.

Preparing the Region for a Backup

When a managed node is installed, a backup object is created in the Tivoli object database. You can view a list of backup objects using the following command:

```
wls -o /Library/BackupClient
```

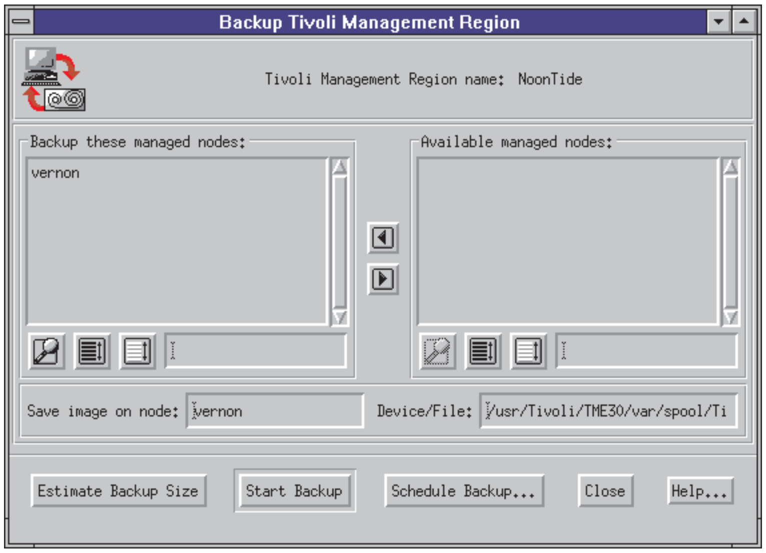
Note that if a backup object is not created for a managed node, this could indicate that the installation did not complete correctly (see "Database cannot be backed up" on page 51).

Periodically repair inconsistent object references within the database before performing a backup. Tivoli Management Framework provides the **wchkdb** command for this purpose. The normal invocation is **wchkdb** –u. For more information about repairing the object database on the Tivoli server, see "Repairing the Tivoli Database" on page 52.

Backing up databases from the desktop

To back up databases on one or more managed nodes in the Tivoli region, follow these steps:

1. Put the Tivoli region in maintenance mode, as described in "Placing a Tivoli region in maintenance mode" on page 2.
2. Select **Backup** from the **Desktop** menu to display the Backup Tivoli Management Region window.



Backup Tivoli Management Region

Tivoli Management Region name: NoonTide

Backup these managed nodes: vernon

Available managed nodes:

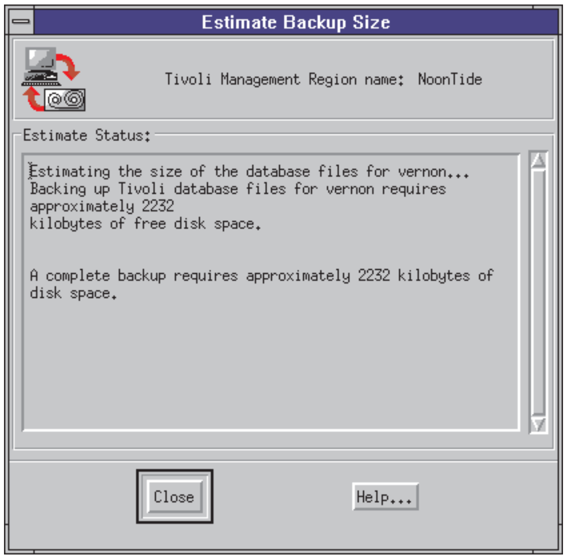
Save image on node: vernon

Device/File: /usr/Tivoli/TME30/var/spool/Ti

3. Select one or more managed nodes from the **Available managed nodes** scrolling list and click the left-arrow button. The selected managed nodes move from the **Available managed nodes** scrolling list to the **Backup these managed nodes** scrolling list. You can select any combination of managed nodes and the Tivoli server to be backed up. If you want to back up the entire Tivoli region, select all managed nodes listed in the **Available managed nodes** scrolling list.
4. Specify the machine on which the backup image or device is located in the **Save image on node** field.

   Note: You cannot store backup images on a NetWare managed node.
5. Specify the device or file name in which the backup is to be saved in the **Device/File** field.

   The default directory to which backup files are written has **root** write permissions only. If you cannot log in as the **root** administrator, you must change the location of the backup file to a directory to which you have write access.
6. Click the **Estimate Backup Size** button . The Estimate Backup Size window is displayed.

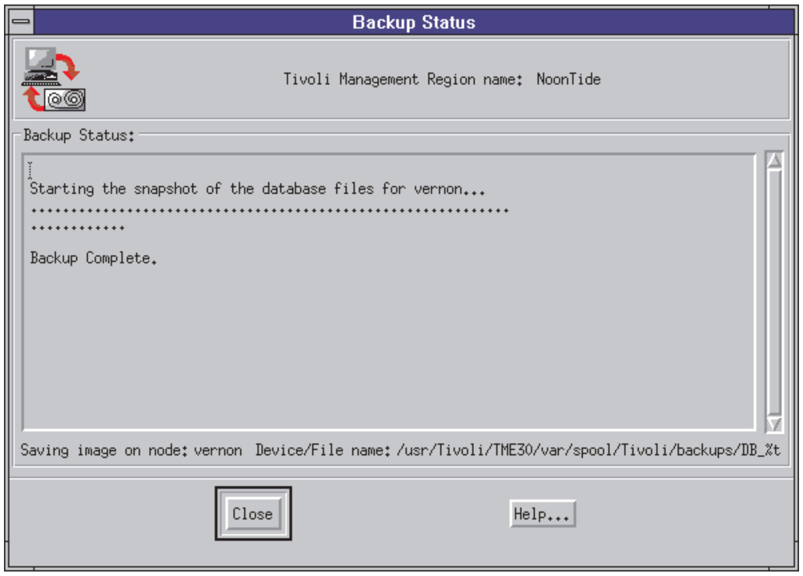


```
Estimating the size of the database files for vernon...
Backing up Tivoli database files for vernon requires
approximately 2232
kilobytes of free disk space.


A complete backup requires approximately 2232 kilobytes of
disk space.
```

The **Estimate Status** section displays the estimated backup size required for each managed node. When the estimate is complete, the Estimate Backup Size window also shows the total amount of space required to back up the selected managed nodes.

7. Click **Close**.
8. On the Backup Tivoli Management Region window, choose one of the following to schedule a backup or start a backup immediately.
   - To schedule a backup for a later time, click the **Schedule Backup** button. For more information about scheduling operations, see the *Tivoli Management Framework User's Guide*.
   - To begin an immediate backup of the selected managed nodes, click the **Start Backup** button. The Backup Status window is displayed and the backup operation begins.



```
Starting the snapshot of the database files for vernon...
............................................................
............

Backup Complete.
```

```
Saving image on node: vernon  Device/File name: /usr/Tivoli/TME30/var/spool/Tivoli/backups/DB_%t
```

   - Click **Close** to close the current window and return to the **Desktop** window.

Backing up databases from the command line

For information about using the command line to back up managed nodes in a Tivoli region, see the **wbkupdb** command in the *Tivoli Management Framework Reference Manual*.

Changing the Default Backup Directory

If you require a different location for the default backup directory, you can set it from the command line. When setting the directory, you also can modify the name of the backup file. Reasons why you might want to change the default backup directory include if you changed the database directory path or if you want a different permanent location for backups, such as a tape device or a separate file system.

The default backup directory is not changed if the database directory changes. The backup process also does not use the **DBDIR** environment variable.

The following table provides the context and authorization role required for this task:

Activity	Context	Required Role
Change the default backup directory	Tivoli region	**super** or **backup**

To change the default backup directory or to determine the current location of the backup directory, follow these steps:

1. Log in to a managed node or the Tivoli server on which your Tivoli administrator has an alias with the **super** or **backup** role for the Tivoli region.
2. Ensure that your system environment is set correctly. For more information, see “Setting Tivoli environment variables” on page 1.
3. To find the backup object ID (OID) class, enter the following command:
   ```
   OID=`wlookup TMRBackup`
   ```
4. To retrieve the existing directory location and path name, enter the following command with the OID that was returned in the previous step:
   ```
   idlcall $OID _get_default_device
   ```
5. To set the default backup directory to a new location, enter the following command:
   ```
   idlcall $OID _set_default_device '"new_path_name"'
   ```

You can have Tivoli Management Framework automatically append the name of the backup file with a date and time of the backup. Use the **%t** variable when setting the directory location. For example, if you specify the following command:

```
idlcall 1264987995.1.351 _set_default_device '"/usr/local/backup/DB_%t"'
```

The resulting backup file name would look similar to DB_Dec21-0955.

Other important methods in the backup class object include **_get_default_host**, which is the host that is backed up if none are specified, and **_get_server_files** and **_get_client_files**, which are the files to be backed up. Do not remove any files from the default server or managed node file list.

Changing the Destination of a Temporary Backup File

During a backup, the backup operation creates a temporary backup file as a staging site while it copies the databases. If you run a backup from the desktop, this temporary file is written to the database directory. If you run a backup from the command line using the **wbkupdb** command, the temporary file is written to the current directory. In both cases the directory in which the temporary file is written should have as much available disk space as the largest compressed database. If the directory does not have enough space, you can change the directory in which this file is created by setting the **TMPDIR**, **TMP**, or **TEMP** variables in either the Tivoli environment or the shell environment.

The following steps describe how to set the **TMPDIR**, **TMP**, and **TEMP** environment variables.

1. To retrieve the current environment settings and write them to a file, enter the following command:
   ```
   odadmin environ get > file_name
   ```
2. To add the following environment setting to the file, enter the following command:
   ```
   TMPDIR=/home/big_dir >> file_name
   ```
3. To write the environment settings back to Tivoli region, enter the following command:
   ```
   odadmin environ set < file_name
   ```

The temporary backup file is deleted when the backup procedure completes.

Note: You also can set and export **TMPDIR**, **TMP**, or **TEMP** environment variables in their command line shell.

Identifying Managed Nodes in a Backup File

The backup file is a tar file of compressed tar files. Sometimes you need to know which managed nodes are in a backup file. Do this in the bash shell or other supported shell using the **tar –t** command.

For example, the following output shows that two managed nodes with host names **rh0255b** and **rh0255a** are backed up:

```
# tar -tvf $DBDIR/../backups/DB_Oct20-1609
-rw-------   0 3  2207243 Oct 20 11:09:44 1997 rh0255b.itsc.ibm.com
-rw-------   0 3    15256 Oct 20 11:09:51 1997 rh0255a
```

Restoring Databases in a Region

If a disk drive on a managed node fails or the file system that stores the information gets corrupted or is lost, the data can be recovered by restoring the Tivoli server or managed nodes from an earlier backup. Tivoli Management Framework usually reports irreparable damage to the object database using messages, such as `Persistent storage failure`. In this instance, there is usually little option but to revert to a backup of the database. You can also use this process when upgrading a managed node or the server to a new version of the operating system that is supported for the Tivoli environment.

When restoring a managed node or server, any changes that occurred since the last backup are lost. After a failure occurs, it is important to have an audit trail to track changes that occur in the Tivoli environment. Examples include the following:

- Endpoints logging in to the Tivoli region for the first time

- Moving endpoints from one Tivoli region to another
- Adding or removing new gateways, managed nodes, or endpoints
- Creating, deleting, and changing subscriptions
- Creating, deleting, and changing policy regions
- Changing Tivoli applications

Thus, it is important to always have a way to track system management tasks performed in the Tivoli environment, and other operations that occur in the Tivoli environment.

Tools that are useful for creating an audit trail include the following:
- Notices provided by Tivoli Management Framework
- Tivoli logs for the endpoint, managed node, gateway, or Tivoli server
- Change request pages
- System-generated event records when policies, scripts, or tasks are run

For example, if you delete endpoints after the last backup, those endpoints still exist in the object database after the restore operation. You can view the epmgrlog file to obtain a list of the endpoints deleted so that you can recreate your changes since the last backup.

Although the information that requires auditing varies according to the Tivoli applications deployed and the architecture of the Tivoli environment, good information to capture is endpoint names or labels, policy regions, and profile names.

The following table provides the context and authorization roles required for this task:

Activity	Context	Required Role
Restore one or more managed nodes, the Tivoli server, or the entire Tivoli region	Tivoli region	**restore** or **super**

You can restore one managed node, several managed nodes, the Tivoli server, or the entire Tivoli region. Because the nature of a restore operation affects the underlying database of a managed node, you can only restore the data from the command line.

Consider the following recommendations while planning your recovery strategy:
- Use the **wchknode** command to synchronize the restored managed node with the Tivoli server database after performing the restore operation.
- During a backup, the **wbkupdb** command also saves any old versions of files and of the notification database (notice.bdb). Normally, these are not restored because you probably do not want to reread old notices. If you want to restore the notices for some reason, such as a problem with the current notices database, you can restore them specifically through command line options. For an example of restoring notices, see “Items Not Restored from a Backup” on page 45.
- For the **wbkupdb** command to restore any database (managed node or Tivoli server), Tivoli Management Framework must be running on the Tivoli server and on the managed node whose database is being restored. In other words, the oserv service must be running successfully on both systems, not just the Tivoli

server. If the object dispatcher that is to be restored is not running (and presumably cannot be run because its database is corrupted or missing), you can extract the database manually and put the files in the correct location in the database directory. This process is known as a rescue operation. For more information, see "Restoring a database when the object dispatcher is down."

- As with the backup, the administrator also requires a valid user login name and a group name for the machine on which the backup file is stored. The administrator also needs read permission for the directory that contains the backup file.
- If you are performing a rescue operation, you must have root access on the machine where the corrupted database is located.

There is a distinction between a standard restore operation and a rescue operation. A restore operation can take place when a Tivoli object dispatcher is running. If the object dispatcher cannot be started on a managed node, there is a rescue procedure to restore the database.

Restoring a database when the object dispatcher is running

The **wbkupdb** command not only backs up but also restores Tivoli databases. You can provide a list of managed node names as options to the **wbkupdb** command. The following command example restores a single managed node, **fuji**. The backup file used to restore the managed node is /usr/backups/TMR1.bk.

```
% wbkupdb -r -d /usr/backups/TMR1.bk fuji
```

For more information about restoring a Tivoli database, see the **wbkupdb** command in the *Tivoli Management Framework Reference Manual*.

Restoring using the **wbkupdb** command with the **–r** option causes the object dispatcher to be reexecuted after the restore operation is complete. The object dispatcher then uses the restored database. Restoring with **–r –R** copies the backup files (***.restore**) to the database directory without restarting the object dispatcher. The change is picked up at the next time the oserv service is started.

Restoring a database when the object dispatcher is down

If the object dispatcher to be restored is not running (presumably because the database is corrupted or missing), you can extract the database manually and restore the files to the correct location in the database directory.

To restore the database for a Tivoli server or managed node on systems other than NetWare systems, follow these steps:

1. Ensure that your system environment is set correctly as described in "Setting Tivoli environment variables" on page 1.
2. To change directories to the database directory, enter one of the following commands:
   - For UNIX operating systems, enter the following command:
     ```
     cd $DBDIR
     ```
   - For Windows operating systems, enter the following command:
     ```
     cd %DBDIR%
     ```
3. To extract the contents of the backup file, enter one of the following commands:
   - For UNIX operating systems, enter the following command:
     ```
     tar xvf /var/spool/Tivoli/db/backups/DB_May02-1106 hostname
     ```
   - For Windows operating systems, enter the following command:

```
tar xvf D:\Tivoli\db\backups\DB_May02-1106 hostname
```

A file called *hostname* is created in the database directory.

4. Uncompress and extract the contents of the backup file with the following command:

   ```
   uncompress -c < hostname | tar xvf -
   ```

5. To remove the extracted backup file, enter the following command:

   ```
   rm hostname
   ```

6. On a UNIX shell, enter the following command to move the restored files to the current naming convention:

   ```
   files=`ls -d *restore | sed `s/.restore//``
   ```

7. Remove database files with a **.old** extension. Then save the current database files by renaming them with **.old**. Restore the database files. For example:

   ```
   for i in $files
   do
   echo moving $i.restore $i
   rm -rf $i.old
   mv $i $i.old
   mv $i.restore $i
   if [$? !=0];then
   echo mv failed.
   exit 1
   fi
   done
   ```

8. To start the object dispatcher, enter the following command, where *num* is an optional dispatcher number. :

   ```
   odadmin start num
   ```

   If you do not specify the *num* option, the object dispatcher is restarted on the local managed node.

   Note: You cannot use the **odadmin** command to start the object dispatcher remotely on NetWare systems. For more information, see "Stopping and starting the object dispatcher" on page 8.

To restore a Tivoli server or managed node on a NetWare system, follow these steps:

1. To stop the object dispatcher on NetWare systems, enter the following command:

   ```
   odadmin shutdown num
   ```

   where *num* is the object dispatcher number.

   Note: If the **odadmin** command is unsuccessful, run the **oservend** command from the NetWare machine.

2. Do one of the following:
   - If you performed the backup on a UNIX operating system, enter the following command, where /tmp/backupNW is the location of the NetWare backup files and a file named hostname is on that directory:

     ```
     tar xvf /tmp/backupNW hostname
     ```

   - If you performed the backup on a Windows operating system, enter the following command, where /tmp/backupNW is the location of the NetWare backup files and a file named hostname is on that directory:

     ```
     tar xvf d:/tmp/backupNW hostname
     ```

3. To uncompress the file, do one of the following:

   - If you performed the backup on a Windows operating system using Requester, enter the following command, where *mapdrive* is the drive that is mapped to the NetWare system:

     ```
     mapdrive:/uncompress -c <hostname
     ```

     For example, if **m**: is the mapped drive, for \\nwserv\sys, enter this command from the m:\tivoli\db directory.
   - If you performed the backup on a supported platform other than Windows, **FTP** to the hostname file from a Windows machine and then enter the **uncompress** command.
4. To restart the object dispatcher, enter the following command:

   ```
   SYS:TIVOLI\BIN\NWR-IX86\BIN\OSERVRUN
   ```

Items Not Restored from a Backup

When restoring from a backup, a few items are not restored:

- Any endpoints that joined the Tivoli region since the last backup are not restored on the Tivoli server
- The notification database where old notices are stored is not restored on the Tivoli server
- The system files stored in the file versioning system are not restored on the managed node
- Tivoli files stored on the endpoint are not restored on the endpoint
- The gateway database on the gateway (gwdb.bdb file) is not restored on the managed node

After a backup, it is likely that new endpoints will log in to the Tivoli region for the first time. These new endpoints are recorded in the endpoint manager database. After the database is restored from the backup, the new endpoints are no longer represented in the endpoint manager database. The endpoints, however, still have the endpoint daemon running and believe that they are part of the Tivoli region. Note that this also applies to managed nodes created after a backup has been performed.

Those endpoints added to the Tivoli region since the last backup are considered orphaned. An endpoint is considered *orphaned* when the endpoints identify themselves as being part of the Tivoli region based on the information in the lcf.dat file; however, the endpoint manager and Tivoli name registry do not recognize that the endpoints ever logged in.

These endpoints remain in the orphan state until the endpoint manager is notified to let the endpoint return. At that point, the endpoint manager creates a new identity for the endpoint. Thereafter, the endpoint begins the login process. Thus, you will not be able to perform any actions on those endpoints, such as software distributions or inventory scans, until it rejoins the Tivoli region. For additional discussion of problems with endpoints, see Chapter 4, "Troubleshooting endpoints," on page 81.

In addition to problems with orphan endpoints, the Tivoli files on the endpoint are not backed up. "Endpoint Recovery" on page 47 discusses strategies for restoring an endpoint after its disk fails or its file system becomes corrupted.

Even though the gateway database on the gateway (gwdb.bdb file) is not part of the backup, it actually does not need to be backed up. The gwdb.bdb file does not contain persistent data. The gateway gets its copy of the database from the endpoint manager when the gateway boots up.

During a backup, the **wbkupdb** command also saves the notification database (notice.bdb). It is not typically restored. In some instances, old notices can be the only way to know what has happened since the last backup. Reading the notices since the last backup provides a history of Tivoli environment operations. This history contains what was occurring to the Tivoli environment and provides clues to what actions caused the Tivoli environment to fail. If you restore the notices from the last backup using the following example, the most recent notice.bdb file is overwritten with the backed up version.

If you want to restore the notices for some reason, such as a problem with the current notices database, you can restore them by completing the steps in the following code example. In this example, the Tivoli server is called **fuji**. The commands must be run at the Tivoli server, because notices are not held on the managed nodes.

```
cd $DBDIR/../backups
ls -l
total 4064
-rw-r--r--   1 root    root   20722064   Nov  26  10:32 DB_Nov26-1032
mkdir /tmp/work
cd /tmp/work
tar -xvf $DBDIR/../backups/DB_Nov26-1032 fuji
mv fuji fuji.Z
uncompress fuji
olddir=`pwd`
cd $DBDIR
tar -xvf $olddir/fuji notice.bdb.restore
tar -xvf $olddir/fuji notice.log.restore
odadmin shutdown
mv notice.bdb.restore notice.bdb
mv notice.log.restore notice.log
odadmin start
```

Any old versions of system files (the file_versions directory) are also not restored. If you want to restore versioned system files, use the same procedure as the one for restoring the notices database. However, when you unpack the tar files, untar the whole file and then move the file_versions directory as necessary. For more information about the file_versions directory and the revision control system for Tivoli Management Framework, see “Deleting versioned system files” on page 30.

Recovery Strategies

This section describes some strategies for restoring the Tivoli environment and the impact to Tivoli operations. Depending on the Tivoli environment, applications installed, and high-availability needs, these suggestions should be incorporated into a plan that best fits your system management requirements. Suggestions for Tivoli server recovery, gateway recovery, managed node recovery, endpoint recovery, and hub-spoke recovery are provided in the following sections. Although individual solutions for recovery of managed node and servers are important, many of the suggestions provided can be scripted in advance to help during a recovery. The output of these scripts should log the changes as they occur to text files stored on a file system other than the ones Tivoli Management Framework is using.

Tivoli Server Recovery

When a Tivoli server has a software or disk failure, this situation prevents management operations to all gateways and endpoints in that Tivoli region. With the Tivoli server unavailable, only local operations at each managed node or gateway are possible, because security authorization provided by the Tivoli server is unavailable. Endpoints, managed nodes, and gateways remain online but will not be able to perform upcalls to the Tivoli server.

During the time that the Tivoli server is unavailable, you also cannot perform resource changes, such as adding or removing a managed node, subscribing endpoints to profiles, or bringing new endpoints in to the Tivoli region. It is important to remember that any events coming from the endpoints could be lost during the Tivoli server downtime.

If the Tivoli server fails, follow this recovery strategy:

1. Restore the disk or database on the Tivoli server from the latest available backups.
2. Start the Tivoli server.
3. Using notices, events in script files, and any external documentation, identify changes to applications, endpoints, or managed nodes since the last backup. Recreate any changes.
4. Monitor endpoints as they log in. They should go through their usual login scripts and subscription relationships.
5. If the Tivoli region is connected to other Tivoli regions, update resources between the regions to resynchronize the shared resources.

Gateway and Managed Node Recovery

When a gateway fails, this isolates all of its assigned endpoints until the endpoints successfully log in to an alternate gateway. Until those endpoints log in to an alternate gateway, management operations against isolated endpoints fail. For more information about the impact of a failed gateway on endpoints, see "Gateway is unavailable during normal login" on page 94. If a managed node fails, no managed node services are accessible for management operations, including gateway, repeater, or distribution services.

If a gateway or managed node fails, follow this recovery strategy:

1. Restore the disk or database from the latest available backups.
2. Start the managed node.
3. Monitor endpoints as they log in. They should go through their usual login scripts and subscription relationships.

Instead of restoring the managed node database, you can delete the gateway and managed node from the Tivoli server. Then reinstall the managed node and gateway and reconfigure the managed node for any custom repeater settings for multiplexed distribution. For more information about installing managed nodes and gateways, see *Tivoli Enterprise Installation Guide*. For more information about configuring repeaters, see the *Tivoli Management Framework User's Guide*.

If a gateway will be out of service for an extended period of time, its endpoints should be reassigned to another gateway. To reassign isolated endpoints, see "Unexpected results with endpoint migration" on page 97.

Endpoint Recovery

In some instances, it is necessary to restore an endpoint if its disk fails or its file system becomes corrupted. In this situation, the endpoint is not available for any management operations. Endpoint recovery is different from the recovery of other elements of a Tivoli environment—the files on the endpoint are not part of the Tivoli backup process. Instead, endpoint recovery depends on file system backups or installing the endpoint again.

If an endpoint fails and you have a full disk backup of the endpoint file system, follow this recovery strategy:

1. Restore the disk from the latest available backup.
2. Restart the endpoint daemon on the endpoint.

If an endpoint fails and you do *not* have a backup of the endpoint file system, follow this recovery strategy:

1. Remove the endpoint, as described in the *Tivoli Enterprise Installation Guide*, if possible.
2. Use the **wdelep** command to delete the endpoint from the Tivoli object database.
3. Reinstall the endpoint.
4. If any additional Tivoli applications are used to manage the endpoint, run those application processes again on the endpoint.

Hub-and-spoke architecture recovery

Some environments have a two-tier system, often referred to as the hub-and-spoke architecture. In this architecture, a master Tivoli server has Tivoli servers connected to it. The master Tivoli region is commonly called the hub. The Tivoli regions connected to the hub Tivoli server are called the spokes. For additional descriptions of the hub-and-spoke architecture, see the *Tivoli Management Framework Planning for Deployment Guide*.

There are additional considerations for recovery in a hub-and-spoke architecture when the hub Tivoli server fails. Although Tivoli operations can continue on the spoke Tivoli regions, the hub region failure means the loss of the global overview of the system. System-wide changes cannot be carried out, and if change is allowed on spoke regions, those changes will not be reflected at the hub level until it is recovered. In addition, managed nodes and endpoints in the spoke region can continue without the hub for some time. Avoid changes at the spoke region level, such as adding new endpoints. Moving an endpoint among gateways within the same spoke region should not present a problem, because it is primarily the spoke endpoint manager performing this work.

If the hub Tivoli server fails, the recovery strategy is as follows:

1. Minimize changes on the spoke Tivoli server during the hub outage. Define profile managers in the spoke Tivoli regions where the subscriptions are located rather than at the hub level. This allows the spoke Tivoli regions more operations while isolated from the failed hub region and simplifies resynchronization after the hub region returns to operations.
2. Restore disks or databases on the hub Tivoli server from the latest available backups.
3. Start the hub Tivoli server.
4. Identify changes since the last backup using the notices database and events from any script logs. Endpoints new to the Tivoli region are reconciled during the resource update. Recreate any changes.
5. Update resources between the hub Tivoli region and each spoke region to resynchronize the shared resources.

Troubleshooting Backup and Restore Operations

This section describes problems you might encounter during backup and restore operations.

Correcting access problems for backups

If you encounter problems while performing a backup, try some of these tasks to ensure that you have the correct access:

- On the computer system where the backup will be stored, create a new backup group of all administrators who will perform backups and assign ownership of the backup directory to that new group.
- Temporarily change the permissions on the backup directory to allow anyone to write to the directory.
- Create a task for administrators with the **backup** role that runs as root and performs the **wbkupdb** command.
- Schedule the backups as the root administrator instead of a non-root administrator.
- Change the user login name and group name to a valid one that has enough permissions to make the backup.
- Be careful when writing backups to a Network File System (NFS). You must consider the security implications of using root on an NFS.

Common errors during backup

The following list details common error messages and provides suggestions for correcting each error.

- **The system clock is older than the timestamps of the database files in $DBDIR.** This situation prevents normal operation of the Tivoli server. To correct this error, shut down the Tivoli server and perform one of the following actions:
  - Reset timestamps on all Tivoli files within the Tivoli installation directory to match the system time. The files under the Tivoli should be older than the system clock time.
  - Change the system clock time to the correct time.
- **Date Time (18): 'iom_open' failed with code '13': A program used to copy the database on the managed node xxxx failed with exit status 1 (iomsend)**

  When the **wbkupdb** command is invoked from a script, it uses the current working directory. This directory must be writable by the **nobody** account.
- **Date Time (18): 'iom_open' failed with code '28': Not enough space.**
  - Check the file space for the database directory.
  - Check the file space for the /tmp or C:\temp directory.
  - Check the file space for the current working directory.
  - Check user permissions to write to the current directory.
- **Date Time (18): 'iom_open' failed with code '30'**

  Check user privileges to write to the current directory.
- **Date Time (18): 'iom_open' failed with code '67': A reference to an object is invalid. The object no longer exists or is in a disconnected TMR.**

  Clean up the backup with the **wchkdb –ux** command.
- **Date Time (18): 'iom_open' failed with code '67': A reference to an object is invalid. The managed node is down or unreachable.**

Verify that network services are operational, and restart the object dispatcher on the managed node.

- **Date Time (18): 'iom_open' failed with code '67': A reference to an object is invalid. The managed node is too busy to extract a consistent backup...**

  To correct this error on a UNIX operating system, check the system date and time. The $DBDIR/file_versions directory and contents have a date and time-stamp problem. Use the **touch** command on the directory.

  To correct this error on a Windows operating system, verify that network services are operational.
- **Date Time (18): 'iom_open' failed with code '67': A reference to an object is invalid. "*no format string*"**
  - The $DBDIR/file_versions directory is missing. Create the directory as root/bin with file permissions 755 on UNIX or Everyone (Full Control) on Windows operating systems.
  - A path name within the **$DBDIR/file_versions** directory is too long. Rename it or delete it.
  - Check the file space and permissions on /tmp, /var/tmp, $DBDIR, or %DBDIR%\tmp.
  - Perform the following steps on the Tivoli server and managed node:
    1. Enter the following command:
       ```
       odadmin environ get > afile
       ```
    2. Edit *afile* and add the line `LANG=C`.
    3. Enter the following command:
       ```
       odadmin environ set < afile
       ```
    4. On AIX operating systems, set LOCALE to C and reboot the system.
    5. On HP-UX operating systems, set LANG to C and reboot the system.
- **Date Time (18): 'iom_open' failed with code '67': A reference to an object is invalid. Unknown internal error: shell method wrote malformed ASCII exception.**
  - The /tmp or C:\temp directory is full.
  - The system is too busy, and the database is updated before a snapshot can be taken of it.
  - Place the Tivoli server in maintenance mode with the **wlocktmr –e** command before backing it up.
  - If the backup is run from a job by the scheduler, increase the timeout value.
- **Date Time (18): 'iom_open' failed with code '67': A reference to an object is invalid. A communications failure occurred: destination dispatcher unavailable.**
  - Verify that network services are operational.
  - Restart the object dispatcher on the managed node.
- **Date Time (18): 'iom_open' failed with code '67': A reference to an object is invalid. Persistent Storage Failures**
  - Check the file space for the database directory.
  - Check the file space for the /tmp or C:\temp directory.
  - Check the file space for the current working directory.
- **"freopen failed with code 13"**

  The user does not have write permissions to the $DBDIR/../ backups directory. The permissions are typically set to **drwxr-xr-x** for root access only.

Database cannot be backed up

There might be occasions when Tivoli Management Framework does not back up the database. This could be caused by an incomplete installation that is not identified as such. The error message you can receive is similar to the following:

```
A nested resource called "BackupClient" nested with a "ManagedNode" called
     information about "everest" was not found.
```

To identify the cause, enter the following command, where everest is the name of the computer system in question:

```
wlookup -ar ManagedNode -n everest
```

The output for a fully installed managed node is similar to the following:

```
TaskExecute   1264987995.2.9#TMF_ManagedNodesureTaskExecute#
BackupClient  1264987995.2.13#TMF_Backup::Client#
imp_TMF_UI::DesktopList 1264987995.2.15#TMF_UI::DesktopList#
imp_TMF_UI::Extd_DesktopList    1264987995.2.28#TMF_UI::Extd_DesktopList#
```

If you are missing any of these entries, such as the **BackupClient** object, the managed node might not have been installed and must be removed and reinstalled. The backup process automatically skips any managed nodes that do not have backup objects.

If you want to check that all managed nodes have backup objects and will, therefore, be available for backups, you can enter the following command to display a list of backup objects:

```
wls -o /Library/BackupClient
```

An example of the output of this command is as follows:

```
1264987995.1.350#TMF_Backup::Client#    stout.tivoli.com
1264987995.2.13#TMF_Backup::Client#     rh0255a
```

When you use this method to list backup objects, the label appended to the object dispatcher of the managed node might not reflect the true name of the server. Thus, you must use the dispatcher number to identify the managed nodes with backup objects, not the label shown to the right of the output. In the example shown, you would use the dispatcher number **1264987995.1.350** to identify the managed node, and not the label **stout.tivoli.com**.

Malformed ASCII Exception

Sometimes the database is too busy to be backed up. This is often characterized by the following error message:

```
Error - Unknown internal error: shell method wrote malformed ASCII exception
```

This message comes from the snapshot program and indicates that the database is being updated too frequently to copy the database before it changes. An error message stating that the system is too busy can also be displayed. To resolve this error, you must either put the Tivoli region in maintenance mode to make the backup or wait until the Tivoli server is not so busy. For more information about how to put the Tivoli region in maintenance mode, see "Placing a Tivoli region in maintenance mode" on page 2.

IOM Route Timeouts

Clients send their backup files to the server through an Inter-Object Messaging (IOM) channel. The managed nodes must be able to contact the server by the IP address used to install the server. If the server does not get a response from the managed node, it generates an IOM timeout communication error.

One common example is if the IP address of the Tivoli server is changed (see “Changing the IP Address of a Tivoli server” on page 17). Also, ensure that there is communication between the server and managed nodes. For information about using the **ping** command by both name and address, see “Viewing the ping behavior of the object dispatcher” on page 13.

Note that you can use the **odadmin** *set_iom_by_name* command to enable managed node communications to rely on the host name rather than the IP address of a Tivoli server when interpreting an IOM key and making a connection. This is useful when you have multihomed servers that are known by different IP addresses on different subnets. For more information about the **odadmin** command, see the *Tivoli Management Framework Reference Manual*.

Repairing the Tivoli Database

Although Tivoli Management Framework uses a distributed transaction service to ensure that modifications to the distributed databases are consistent and complete, problems can occur in certain circumstances. For example, a disk drive crash or a corrupted file system can result in inconsistent and incorrect data in the management database.

To resolve such problems, Tivoli Management Framework provides the **wchkdb** and **wchknode** commands, described in the following sections.

The following table provides the context and authorization role required for these tasks.

Activity	Context	Required Role
Check the database consistency of one or more resources or the entire region (**wchkdb**)	Tivoli region	**senior** or **super**
Check the database integrity after the database for a managed node is restored (**wchknode**)	Tivoli region	**senior** or **super**

Because the nature of checking database integrity affects the underlying databases, you can perform these operations only from the command line. For more information, see the **wchkdb** and **wchknode** commands in the *Tivoli Management Framework Reference Manual*.

Checking Database Consistency

The terms corruption and consistency should be understood in the context of the Tivoli database. When a database is *corrupted*, a problem with the structural integrity of the database makes it incapable of properly storing data. When a database is inconsistent, it can store data properly, but a problem with the data exists.

The **wchkdb** command checks the consistency of Tivoli objects and their relationships, and verifies references from one object to another. It does not check the structure of the database, nor does it check for valid data in the database.

If a reference is not correct, the command attempts to rectify the situation (using the **–u** option) or removes the member when it determines that the member is not a valid entry. This command never deletes an object in the database, nor does it affect any system files. It only modifies resources within the Tivoli environment and database to preserve database integrity.

In the same way that using the UNIX **fsck** command is best with a quiet file system, you should not run the **wchkdb** command while performing arduous operations that manipulate the data in the database, such as large distributions or updates of resources between interconnections, which might cause long-running transactions. Do not perform actions with Tivoli Management Framework that create objects in the database while a **wchkdb** command is running.

Note that the **wchkdb** command cannot repair all database inconsistencies. For example, you cannot use the **wchkdb** command to correct file corruption resulting from power failures. The best protection against these failures is to frequently back up the database so that you always have the ability to restore it.

Note: The **wchkdb** command does not check the gateway database. To check the gateway database, use the **wgateway –dbcheck** command.

You should run the **wchkdb** command in the following situations:
- Before and after you perform an upgrade, install a large patch, or install additional products
- When you remove a machine or group of machines from a Tivoli region
- When you start seeing unusual error messages, such as `object identifier not valid`
- Whenever a managed node fails to install repeatedly
- Whenever you experience a drive crash, power loss, or corrupted data on the disk that contains the Tivoli region database
- Before you perform a backup
- When you restore a database or set of databases from a backup
- When operations such as removing managed resources or applications from the object database fail
- When an incomplete or erroneous database update operation occurs
- When an I/O failure on a managed node occurs
- When installation of an application is incomplete

After restoring Tivoli databases, first run the **wchkdb** command only on the local Tivoli region. After running the **wchkdb** command on the local region, you can run it on all connected regions. Using this order, you ensure that all local data is consistent before you attempt to reconcile any inter-region discrepancies.

When running the **wchkdb** command, consider these precautions:
- Perform database consistency checks regularly, especially prior to performing a backup. This prevents any correctable errors from being stored in the database backup.
- Run the **wchkdb** command each time you delete a registered resource. Deletions are the most common cause of database inconsistencies and corruption.
- Run the **wchkdb** command when no other major Tivoli processes are running.
- Separate the verification and repair operations (since database checking and repair can take a long time in a Tivoli region with a large number of resources). The verification operation informs you of the objects whose database verification failed. The **wchkdb –u** command operation repairs these objects if any errors are returned.

Checking object ID references on a managed node

Checking the consistency of a large Tivoli region (for instance, 200 managed nodes) can take as long as 30 hours, depending on the complexity of the database. This makes it impractical to check the consistency of the database for every backup. In these cases, the **wchknode** command proves useful.

The **wchknode** command is a complementary command to the **wchkdb** (check database) and **wrmnode** (remove managed node) commands. It verifies and updates references to a specific dispatcher number from parts of the Tivoli database and is particularly useful in the following situations:

- After a general failure when removing a managed node
- After completing a **wrmnode** command process for a managed node from the environment
- After restoring a database on a managed node
- After disconnecting Tivoli regions

Unlike the **wchkdb** command, which verifies the integrity of the structure of all objects in the database, the **wchknode** command simply checks that all objects respond to an **objcall** command to get their label. Running this command has less impact on a Tivoli server compared to the impact of some invocations of a **wchkdb** command in large, complex environments. In addition, the command usually finishes within 5 minutes and should not present any difficulties.

Note that neither the **wchkdb** nor the **wchknode** command deletes objects from the database. The **wchkdb** command is focused on fixing references to objects in the form of memberships, subscriptions and back references. It does not delete objects. In some situations the **wchkdb** puts an object in the lost-and-found collection if it is missing some important information. The **wchknode** command simply tests to see if a referenced object ID exists. If it does not exist, it removes the object reference.

When Database Checks Take Too Long or Do Not Complete

The reason database checks can take a long time is directly related to the way the **wchkdb** command works. Running this command involves a serial process that checks a single object across all nodes in the environment.

To understand why a database check can take a long time, you should perform this check in a test environment, and, at the same time issue the **odstat –c** command repeatedly to see what is occurring during a database check. Output similar to the following is displayed:

```
16780 O+hdq           run   0  0  14:03:28
1902971651.1.569#TMF_UI::Presentation# check_db
16781 O+hdq1-16780    run   0  0  14:03:28 redirect check_db
```

While the database is running, and as each part completes, you see output similar to the following occurring repeatedly:

```
16061 O+hdq1-16060  done  18 0 09:27:48  redirect check_db
16062 O+hdq1-16061  done  18 0 09:27:48  redirect check_db
16064 O+hdoq        done  18 0 09:27:48  1902971651.1.365 check_db
```

As an indicator, if checking a single object takes more than an hour, the database check might be hung. Database checks can take a long time if the object being checked is a remote node whose object dispatcher is failing to respond appropriately. In this case, the following information might be helpful in identifying the problem.

When an object despatcher sends a message to a managed node, the response arrives immediately if that managed node is running. If the managed node is down, it could take a long time for the object dispatcher to make that determination because of the way Transmission Control Protocol (TCP) works. Depending on the operating system configuration, TCP could take a long time to timeout; for example, default configuration on Solaris takes 12 minutes. The application is notified by TCP that the managed node is down only after the TCP timeout occurs.

When the **wchkdb** command runs, the object dispatcher needs to contact each managed node for each object. If a managed node is down and if the object dispatcher has to timeout for each object, the **wchkdb** command process could take a long time to finish. But this behavior can be changed.

The **odadmin** command includes an option called **set_keep_alive**, which sets options for watching hosts that are down. When this option is set to **on**, the object dispatcher caches the state of the managed nodes. If the object dispatcher determines that a managed node is down, it keeps that information cached, and does not timeout on that managed node subsequently. If this option is set to **off**, the object dispatcher contacts the managed node for each object, even if the previous attempt timed out. By default, this option is **on**.

If you suspect that the **wchkdb** command is taking too long, run the **odadmin odinfo** command to verify that the **set_keep_alive** option is set to **on**.

If the **set_keep_alive option** is **on**, the following line is displayed:

```
State flags in use = TRUE
```

If the **set_keep_alive** option is **off**, the following line is displayed:

```
State flags in use = FALSE
```

This procedure generally clears the failing object dispatcher and allows the entire database check to finish. If a reboot is not possible or desired, you can reexecute the object dispatcher process on the node each time you note that the check is hung (as it reaches that node during each serial check of objects).

Chapter 3. Troubleshooting the Tivoli environment

In problem-solving situations, you need to understand the interactions of the product components, how to interpret messages and trace output, and what actions you can take to resolve a problem.

Tivoli Management Framework provides a number of log files and tracing utilities that enable you to determine the source of problems when they occur. This chapter describes Tivoli log files, troubleshooting commands, and tips for working with the object database. It also provides a comprehensive troubleshooting procedure, which you can use whenever you begin troubleshooting systems.

This chapter contains the following sections:

Attention:: Most activities described in this chapter do not involve changes to the object database. However, direct object invocations, Interface Definition Language (IDL) calls, and attribute changes in the Tivoli object database have the potential to cause unpredictable results and possibly the complete failure of your Tivoli region. Recovery, if at all possible, usually involves a restoration of the object database. Generally support personnel cannot assist in rectifying such changes. Therefore, back up your object database before performing any direct manipulation, test your changes on an isolated test Tivoli region, and keep a log of every action performed.

General Troubleshooting Procedure

If you are gathering information for support, use this list of steps whenever you begin to troubleshoot your Tivoli environment.

To assess the situation and re-create the problem, follow these steps:

1. To obtain information about the connected managed nodes, enter the following command:

   ```
   odadmin odlist
   ```

   The output shows you the status of the managed node, its IP address, the port, and the alias for the host name. It also includes flags which indicate the status of the node in the Tivoli environment. For more information about the **odadmin** command and **odlist** option, see "Listing active managed nodes" on page 7.
2. To obtain information on the local object dispatcher, including port range restrictions, enter the following command:

   ```
   odadmin
   ```

   For more information about the **odadmin** command, , see "Viewing data for a Tivoli region" on page 4.

3. Determine the method environment that the object dispatcher is using. The method environment is the environment variables that are set inside the process of the method when the method executes. To determine the method environment that the object dispatcher is using, enter the following command:
   ```
   odadmin environ get
   ```
4. To determine which machine is causing the problem, run the **odstat** command, and then review the parent thread IDs in the output and examine suspect machines. The parent thread ID is the dispatcher number of the machines within the Tivoli region. For more information about the **odstat** command, see "Listing the Status of Current and Recent Object Calls" on page 64.

To gather data from each machine where you suspect there are problems, follow these steps:

1. Log on as administrator or as a Tivoli administrator. This helps ensure you are not experiencing authority problems.
2. To enable tracing for error conditions on the current object dispatcher, enter the following command:
   ```
   odadmin trace errors
   ```
3. To enable tracing for all object calls on the current object dispatcher, enter the following command:
   ```
   odadmin trace objcalls
   ```
4. To enable tracing for object service calls on the current object dispatcher, enter the following command:
   ```
   odadmin trace services
   ```
5. Re-create the problem.

To assess the situation on the Tivoli region, follow these steps:

1. To see what operations the object dispatcher is currently running, enter the following command:
   ```
   odstat -c
   ```
   Note: To obtain process IDs, use the **odstat –cv** command.
2. Enter the following command:
   ```
   odstat >odstat.txt
   ```
3. To flush the database files for the current object dispatcher, enter the following command:
   ```
   odadmin db_sync
   ```
4. To collect information to debug methods, enter the following command:
   ```
   wtrace -jHk $DBDIR >wtrace.txt
   ```
5. Collect the text files from steps 2 and 3, the oservlog file, and any useful system logs to use in analyzing the problems.
6. To revert to the default of only logging errors, enter the following command:
   ```
   odadmin trace errors
   ```

To analyze the output from these steps, see the logs and commands described in the following sections. For detailed command descriptions, see the *Tivoli Management Framework Reference Manual*.

Using Log Files

Tivoli Management Framework provides several logs that are generated in response to system activities and resources in the Tivoli environment. By viewing these logs, you can begin to identify when, where, and why certain problems are occurring. These logs can help you pinpoint if there is a problem in the Tivoli environment, if the operating system is causing problems, or if there is a user error. These logs can provide a large amount of detail including listing each method called by applications in your Tivoli environment. Logs include the following data:

- The administrator login under which an operation was performed
- The time the application failed
- Which component of the Tivoli environment failed
- Communication failures and timeouts
- The configuration of the Tivoli environment
- Endpoint login process information
- Database inconsistencies
- Transaction process problems

These logs can be found in the database directory that is located through the following variable names:

- On UNIX operating systems, $DBDIR
- On OS/2® and Windows operating systems, %DBDIR%

Tivoli Management Framework provides these log files (grouped in categories):

- Historic text files, which include the following:

  epmgrlog
  The endpoint manager log that is found only on the Tivoli server

  gatelog
  The gateway log that is found only on managed nodes that are gateways

  oservlog
  The error log of the object dispatcher that is found on each managed node and Tivoli server

- Binary transaction files, which include the following:

  gw_oid.**log**
  The Tivoli gateway database transaction log that is found in the epmgr.bdb directory on the Tivoli server

  notice.log
  The Tivoli notice database transaction log that is found only on the Tivoli server

  odb.log
  The Tivoli database transaction log that is found on each managed node and Tivoli server

- A binary file, which includes the following:

  odtrace.log
  The file that the **wtrace** command reads and translates that is found on each managed node and Tivoli server

- A truncating text file on an endpoint, which includes the following:

lcfd.log
The endpoint log file that is found in the /Tivoli/lcf/dat directory on endpoints only

During installations, Tivoli performs default logging in the following locations:
- On UNIX operating systems, /tmp
- On OS/2 and Windows operating systems, %DBDIR%\tmp

For Novell NetWare servers, no log file is created when you install. The only information available is the sys:/etc/console.log file, which displays operating system and gateway activities and the output of the **nw_TMR_install.sh** script on the Tivoli server. To start the log file, use the **load conlog** command. For more installation information, see the *Tivoli Enterprise Installation Guide*.

Some log files are reset each time the process is restarted; other log files have information appended to them and can grow indefinitely. Therefore, monitor these log files for growth and truncate or archive them on a regular basis. When truncating or archiving log files, you should shut down the process that is writing to the log file, move or rename the log file, and then restart the process. The process will create a new log file when it starts. By moving or renaming the log file, you can delete or archive the file at a later time. Examples of log files that log indefinitely are: epmgrlog, gatelog, and oservlog.

The following sections describe the Tivoli logs and the type of information contained within each log.

epmgrlog

The epmgrlog file is a text file that records status information about the endpoint manager and transactions between the endpoint manager, gateway, and endpoint. You can find useful information in the epmgrlog file, such as endpoint login information and gateway assignments. Other types of information often found in this file include the following:
- IP addresses and, potentially, object dispatcher numbers of endpoints
- The gateway object identifier (OID) and IP address of the intercepting gateway during login
- Initial, migration, and isolation endpoint logins
- Errors during endpoint logins
- Additions and deletions of endpoints

The epmgrlog file is located in the /db directory on the Tivoli server. This file grows indefinitely unless modified. Truncate the epmgrlog file regularly.

gatelog

The gatelog file is a text file that records status information about the gateway and the transactions between the endpoint manager, gateway, and endpoint. The gatelog file is useful for debugging problems with endpoint login, distributions to endpoints, monitoring of endpoints, and other endpoint activities.

The gatelog file is located in the /db directory on the computer that hosts the gateway. This text file exists on any managed node that is defined as a gateway, including the Tivoli server.

The amount of information logged in this file depends on the logging level set by the **wgateway set_debug_level** command. For more information about how to set the logging level for the gatelog file, see the **wgateway** manual page. This file grows indefinitely unless modified. It is recommended that you truncate the gatelog file regularly.

<gw_oid>.log, notice.log, and odb.log

The endpoint manager, notices, and object dispatchers use the *gw_oid*.log, notice.log, and odb.log files, respectively. Do not erase these binary transaction files because the files can contain transaction data. These three files enable you to roll back database transactions on the gwdb.bdb, notice.bdb, and odb.bdb database files in the event that a transaction failed or was stopped. They can increase in size while transactions are still pending, but after transactions are completed or committed, they should return to zero. For more information about transaction troubleshooting, see “Displaying the Status of Current Transactions and Locks” on page 70.

An **odadmin db_sync** command flushes the object database state, which should zero the log file. This synchronization process occurs automatically in the object dispatcher every three minutes.

You can use the **logls** command to output a readable version of a transaction log file. It is primarily a diagnostic tool for debugging the transaction manager and assists you by tailoring the information displayed from that file. Some of the command options provide the ability to view only log headers, to retrieve a maximum amount of data from the file, or to view only “old pages” log records.

Working with transactions is not a common problem determination technique, so the output will not be meaningful to you unless you are already familiar with Tivoli transactions.

The *gw_oid*.log file is located in the epmgr.bdb directory on the Tivoli server and a cached copy of the file resides on the gateways.

It is important to back up the odb.log file on a regular basis using the **wbkupdb** command. If this task is not performed, incomplete transactions will be left in the database, resulting in an inconsistent database on restore.

Attention:: Do not modify Tivoli transactions as a routine method of problem determination. Altering transaction data can have serious and irreversible effects on the operation of a Tivoli region. It is recommended that you use this information to find out more about the status of the Tivoli region and to use other methods to attempt to rectify problems.

odtrace.log

The odtrace.log file is composed of a log of object dispatcher errors, object method invocations, and object dispatcher services. To set the maximum size of the odtrace.log file, use the **oserv –t** command. In addition, it is important to note that the odtrace.log file cannot be read by normal editors; it is parsed by the **wtrace** command. For more information about the odtrace.log file, see “Retrieving Information to Debug Methods” on page 67.

oservlog

Each object dispatcher process maintains an error log that is located in the Tivoli database directory. The oservlog file provides information about the object dispatcher, such as when it started, when it shut down, and what error and information messages were displayed.

Some entries in the oservlog file are passed to syslogd (the system log facility). You can use the **syslogd** command to collect oservlog messages from different systems and store the messages in a single location. You also can use the **syslogd** command to filter messages below a specified priority level. The object dispatcher service uses the syslog facility. Refer to the **syslogd** and **syslog.conf** manual pages for information about how to collect and filter oservlog messages.

You can use the oservlog file to help locate errors. Some entries in the oservlog file are time stamped. If an error occurs, you should look for an error message with a time stamp close to the time the error occurred.

Note: Windows operating systems do not have a syslog facility. For Tivoli Management Framework, you must read the object dispatcher error log from the oservlog file.

You can use the **syslogd** command to redirect messages of a specified severity to terminals or files that you set up. The object dispatcher is used when updating the syslog file.

The oservlog file uses the following symbols:

!	Logged to **syslog** as **LOG_ALERT**. These messages usually indicate that an object dispatcher is stopping abnormally.
@	Logged to **syslog** as **LOG_CRIT**. These messages indicate errors based on external dependencies. You should be able to fix these types of problems, for example, a file was not found.
#	Logged to **syslog** as **LOG_ERR**. These messages indicate less severe errors or errors that you might not be able to correct, for example, a method core dumped.
$	Logged to **syslog** as **LOG_NOTICE**. These messages report notable changes in the Tivoli environment, for example, the IP address of a Tivoli server was changed.
^	Logged to **syslog** as **LOG_DEBUG**. These messages are used for debugging. You can use these messages if you are using Application Development Environment (ADE) and are using the **wdebug** or **wdbg** facility.

In addition to these errors, the object dispatcher shutdown messages are also forwarded to the syslog file.

The oservlog file grows indefinitely unless modified. Truncate the oservlog file regularly.

Note: For descriptions of common oservlog error messages, see “Tivoli Object Dispatcher (oserv)” on page 111.

lcfd.log

The lcfd.log file is located in the /Tivoli/lcf/dat/1 directory on the computer that hosts the endpoint. This file contains information about endpoint logins and any upcall or downcall methods to or from the gateway. The amount of data recorded depends on the logging or debug level. The lcfd.log file grows until the endpoint daemon is restarted, at which point the information stored in the lcfd.log file overwrites existing information in the lcfd.bk file, and a new lcfd.log file is created.

The following information is recorded in the lcfd.log file:

- Gateway IP/IPX address and the IP/IPX address of the endpoint as the endpoint attempts to log in to a gateway
- Endpoint dispatcher number, which is useful when there are multiple endpoints on a computer
- Timing of endpoint upcalls and downcalls from applications
- Protocol used for the communication between the gateway and endpoint

You can change the debug level during startup of the endpoint with the **lcfd –d** option. For more information about the **lcfd** command, see the *Tivoli Management Framework Reference Manual*.

Commands for Determining Problems

Tivoli Management Framework provides commands to help you in troubleshooting systems. The most commonly used commands are as follows:

odadmin
Lists the managed nodes in a Tivoli region and configures different aspects of how the object dispatcher communicates, such as defining IP addresses and interconnected regions.

odstat Lists currently running methods and method histories.

wtrace Parses the odtrace.log file, which is created when tracing object calls, services, or errors (tracing is invoked with **odadmin** trace options).

Also described in this chapter are the commands to review and modify transaction status:

tmcmd
Forces a specified transaction or subtransaction to a particular state.

tmstat Displays the status of current transactions and locks.

Managing the Object Dispatcher

The **odadmin** command is the object dispatcher administration interface. This command provides configuration information about the server and allows you to change this information.

The following table provides the context and authorization roles required for this command:

Activity	**Context**	**Required Role**
View and change configuration information about the object dispatcher.	Tivoli region	super, senior

For information about how to use the **odadmin** command to start and stop the object dispatcher, see "Stopping and starting the object dispatcher" on page 8. For more information about the **odadmin** command, see the *Tivoli Management Framework Reference Manual*.

If you enter the **odadmin** command without options, output similar to the following is displayed for the local object dispatcher:

```
Region = 1264987995
Dispatcher = 1
Interpreter type = w32-ix86
Database directory = C:\Tivoli\db\aliburdi.db
Install directory = C:\Tivoli\bin
Inter-dispatcher encryption level = simple
Kerberos in use = FALSE
Remote client login allowed = TRUE
Force socket bind to a single address = FALSE
Perform local hostname lookup for IOM connections = FALSE
Use Single Port BDT = FALSE
Use communication channel check = FALSE
Communication check timeout = default (180 secs)
Communication check response timeout = default (180 secs)
Oserv connection validation timeout = 0
Port range = (not restricted)
State flags in use = TRUE
State checking in use = TRUE
State checking every 180 seconds
Dynamic IP addressing allowed = FALSE
Transaction manager will retry messages 4 times.
```

For a description of **odadmin** output, see "Viewing data for a Tivoli region" on page 4.

For obtaining information about connected managed nodes, you can use the **odadmin odlist** command. This command shows you the status of the managed node, its IP address, the port, and the host name alias. The **odadmin odlist** command also includes values indicating the status of its Tivoli region connection. For more information about the **odadmin** command and **odlist** option, see "Listing active managed nodes" on page 7.

Listing the Status of Current and Recent Object Calls

Tivoli Management Framework provides the **odstat** command, which displays the currently running operations and the previous 200 operations for a specific managed node. Your support provider might ask you to run this command and send its output to assist in problem diagnosis. You can use the **wtrace** command to retrieve all data contained within an **odstat** output. Using the **odstat** command, however, is a much more explicit way to view this data. Use the **wtrace** output when you understand what is going on through examining the **odstat** output.

The following is an example output of the **odstat** command:

tid	type	ptid	state	std O	std E	start	err	Method
2	O+ bhdoq		run	0	0	Tue 11:57		1366474158.1.158#TMF Scheduler:: scheduler#start
3	O +bhdoqs		run	0	0	Tue 11:57		1366474158.1.508#TMF_LCF::EpMgr#epmgr_boot
6	O+ bhdoqs		run	0	0	Tue 11:57		1366474158.1.583#TMF_Gateway::Gateway #gateway_boot

tid	type	ptid	state	std O	std E	start	err	Method
51	O+ hdoqs		run	0	0	Fri 11:41		1366474158.1.528#TMF_UI::Extnd_Desktop#uiserver
54	O+ hdoq	1-51	run	0	0	13:18:04		1366474158.1.179#TMF_Administrator:: Configuration_GUI#launch
167	O+ hdoq	1-51	run	0	0	13:18:06		1366474158.1.196#TMF_PolicyRegion::GUI#launch
307	O+ hdoqs		run	0	0	13:19:47		1366474158.1.643#TMF_UI::Extnd_Desktop#uiserver
310	O+ hdoq	1-307	run	0	0	13:20:08		1366474158.1.641#TMF_Administrator:: Configuration_GUI#launch
400	O+ hdoq	1-51	run	0	0	13:20:10		1366474158.1.644#TMF_CCMS:: ProfileManager#launch
410	O+		run	0	0	13:22:32		1366474158.1.2#query odstat

The output contains the following information:

tid Specifies the thread ID. When you start an object call, two threads are generated: one for the object call itself and one for the method being invoked.

type Lists the thread type. The thread type flags include the following:

O Object call thread (attached to an object request), indicating that the method was invoked here but is running elsewhere.

M Method thread. The object call occurred on a different system, but the object is located on the local system.

O+ Object call and method threads are the same, indicating that the caller and method are both local.

Method type flags include the following:

a	Asynchronous object call
q	Queueing method
o	Per-object method
b	One-way object call
h	Helperless method
d	Daemon method

ptid Lists the parent thread ID or the thread ID of the object call whose method made the current object call. If this field is blank, the object call is external. The number before the dash (-) is the dispatcher number where the parent thread resides. The number after the dash is the thread ID in the parent object dispatcher.

state Lists the following states of the object call thread:

init The thread has been initialized.

ali The thread is performing a lookup on the Tivoli server object database.

mwait The thread is waiting for the associated method thread to complete.

rwait	The thread is waiting for the caller to collect the results of an asynchronous object call.
done	The object call is complete.
coord	The method is serving as a transaction coordinator.
err	An internal error terminated the thread.

Method thread states include the following:

init	The thread has been initialized.
gmeth	The thread is obtaining the method code from another object dispatcher.
hdwt	The thread is waiting for the daemon-method process (the nonqueueing daemon) to be ready to accept another request.
run	The method is running.
serv	The thread is performing an object services call.
done	The method is complete.
twait	The method is waiting on transaction status to commit or abort.

std O Lists the number of bytes written to standard output.

std E Lists the number of bytes written to standard error by the method. Most threads do not write to standard error.

start Lists the time that the thread started. Entries can be one of the following depending on the age of the thread:
- Time–The thread started on this day.
- Day and Time–The thread started before this day, but within in the current week.
- Month and day (UNIX operating systems only)–The thread started before the current week.
- Month (Windows operating systems only)–The thread started before the current week.

err Lists the error status of the thread. If this field is blank, no error occurred.

Error types include the following:

e=n	The method returned *n* as its exit code. Error codes 0 through 21 are reserved for system-defined errors. Tivoli error codes start with 22.
s=n	The method failed due to signal *n*.
S=n	The method failed due to signal *n* and produced a core file. Contact your support provider if you want to debug using this core file.
XXX	An uppercase word indicates an error in the object dispatcher.

Exit codes (**e=**) can come from the system on which the Tivoli desktop is running, Tivoli Management Framework, or an application. It might be necessary to look at different sets of error documentation to find out which is the most likely source. You might be able to use system documentation to obtain help regarding system-produced errors. Most UNIX operating systems include an error file that lists the system error codes and often a short description. On OS/2 operating systems, enter **help** *n*, where *n* is the number of the error message (for example, `help 5`). On Windows operating systems, enter the **net helpmsg** *n* command (for example, `net helpmsg 5`).

method
Lists the text of the method invocation. The first value is the object ID of the object in which context the method was invoked. The format is either three numbers separated by periods or three numbers separated by periods followed by a pound sign (#), the class name, and another pound sign. The next entry is the name of the method followed by its options.

The following is an example of a method listed in the **odstat** output.

```
1242184237.1.516#TMF_SysAdmin::InstanceManager# _get_interfaces
```

Any action that you perform can generate several methods. Use the thread and parent thread ID relationship to determine the order in which the methods started. This is important because the Tivoli environment is multitasking and several administrators might be performing tasks from several different locations. Looking at the thread IDs helps you determine which processes are yours and where an error occurred. When tracking errors, read the odstat file from the bottom to the top and use the thread and parent thread IDs to determine which action caused the problem.

The **odstat** output is listed by order of time and thread. Only the first 80 characters of the method invocation are displayed. IDL-generated method options are not viewable with **odstat**. For more information about IDL-generated methods, see the **wtrace** command in the *Tivoli Management Framework Reference Manual*.

Retrieving Information to Debug Methods

Tivoli Management Framework supports an object tracing mode that records in a log file either all operations or only those that fail. You can use this file later to diagnose problems. This file provides very detailed information and should be used in conjunction with an odstat listing. You can find an error in an odstat listing and use the thread ID associated with it to find additional information in a wtrace file.

The **wtrace** command reads the file in the database directory, called odtrace.log, that was generated by the object dispatcher. This file is static and 1 MB in size. It resides in the Tivoli database directory $DBDIR (%DBDIR% for Windows or OS/2). You can use the **–t** option on the **oserv** command to change the trace file size. However, you must manually start the object dispatcher to specify this option.

Information is written to the odtrace.log file and parsed with the **wtrace** command. This is important to remember because, unlike the **odstat** command, the **wtrace** command can provide debugging information if the object dispatcher is down or has been recycled.

Use the **wtrace** command to interpret trace log records and print the interpreted information in a user-friendly format. The trace log is a circular buffer that contains (by default) the last 512 trace records. A trace record can be either an operation request or an error from an operation request. Your support provider might ask you to enable tracing or print the trace output and send it to assist in problem diagnosis.

The **wtrace** output contains everything that is also found in **odstat** output; however, it is quite detailed and can be hard to read. Having corresponding **odstat** output can make reading the **wtrace** output much easier. For more information, see "Listing the Status of Current and Recent Object Calls" on page 64.

By default, only errors are saved in the odtrace.log file. If you want to save more tracing information, use the **odadmin trace** command with either the **objcalls** or **services** option. Tracing all object operations and services reduces the performance of the Tivoli service. Only enable such tracing for limited periods of time as you troubleshoot systems.

The rate at which **odadmin** data is written to the file depends on the the trace level of the object dispatcher. Various types of methods can be tracked by the object dispatcher in the trace set with the **odadmin trace** command:

errors Records errors in the oservlog file. This is the default setting.

objcalls
: Usual setting to use for problem determination. Traces object method invocation.

services
: Tracks authentication, location, and inheritance (ALI) services requested of the Tivoli server. This tracking quickly fills a trace file.

How to Use Tracing

If you are having problems that are not easily resolved by the messages Tivoli Management Framework provides, follow these steps:

1. To set the tracing option, use the following options (in the following order):
    a. **odadmin trace errors**: Turns on error tracing (recommended).
    b. **odadmin trace objcalls**: Turns on object call tracing (recommended).
    c. **odadmin trace services**: Turns on service tracing (optional).
2. Run the **odstat** command. This helps separate the methods that are involved in the problem by inserting two object calls into the trace (as shown). After you are familiar with reviewing traces, you might want to skip this step and rely solely on finding process IDs or some other method.
    ```
    355 O+  done   15   0 23:53:56    0.0.0 get_oserv
    356 O+  done   2103  0 23:53:58    1480943038.1.2 query odstat
    ```
3. Re-create the problem.
4. Enter the following command, where *odstat_output* is the name of the file containing output from the **odstat** command:
    ```
    odstat -v> /path/odstat_output
    ```
5. Enter one of the following commands:
    ```
    wtrace -jk $DBDIR > /path/wtrace_output
    wtrace -jk $DBDIR >> odstat_output
    ```
    Optionally, use **–jHk** if the trace mentions binary hex data to get the hex data output with an attempted ASCII translation.
6. If needed, refer to the oservlog file.
7. Turn off the trace with the following command:
    ```
    odadmin trace off
    ```
8. Turn on the default tracing with the following command:
    ```
    odadmin trace errors
    ```
9. Review the *odstat_output* file you created in step 4 and begin analyzing the output starting with the **query odstat** line, as illustrated in step 2.

Note: After investigating a system, you should always return to tracing errors only. Leaving other tracing enabled has a negative performance impact. For example, each time a single method is invoked, an average of 3 to 5 service calls are made. This introduces an impact on the object dispatcher because the object dispatcher has to record each step.

The following scenario shows **odstat** and **wtrace** examples of a managed node subscription to a profile manager. For each record in the **wtrace** output, the first line contains the method thread ID, the thread and method flags, the parent thread ID, if one exists, and any error codes. Subsequent lines are similar to the rest of the data in an **odstat** output as shown. Also provided in the trace data is the name of the Tivoli principal that effectively called the method and the path of the method executable.

The **odstat** output includes the following:

```
347 O+ahdoq  1-342  done     9     0 23:53:50 \
     1480943038.1.348#TMF_ManagedNode::Managed_Node# add_subscription
```

The **wtrace** output includes the following:

```
loc-ic  347  M-hdoq    1-342      153
      Time run:    [Thu 01-Jun 23:53:50]
      Object ID:   1480943038.1.348#TMF_ManagedNode::Managed_Node#
      Method:      add_subscription
      Principal:   root@spot (0/60001)
      Path:        /solaris2/TAS/CCMS/profile_organizer
       Trans Id:
         {
           1480943038:1,1480943038:1,4:190
         },
         {
           1480943038:1,1480943038:1,4:192
         }
         #3
  Input Data: (encoded):
         {
           {
             {
               "1480943038.1.580#TMF_CCMS::ProfileManager#" "SwPacks"
             }
             "OBJECT_NIL" "" ""
           }
           0 false
           {
             0
           }
         }
```

In the **wtrace** output, you can find the type of data block being viewed. The **loc** line element refers to a local method. The **rem** line element refers to a remote method. These elements include the following type codes:

ic	Input object call
is	Input service call
oc	Output object call
os	Output service call

After you have set up tracing and generated the output, you might need to investigate a method listed in the output. To find a method, follow these steps:

1. From the **odstat** output, identify the thread ID (tid) where the error occurred. For more information about thread IDs, see "Listing the Status of Current and Recent Object Calls" on page 64.
2. Find the same thread ID in the **wtrace** output.
3. Identify the method, input data, and results for that method.
4. You can then identify the principal that ran the method, for example:

   ```
   Principal:   root@spot (0/60001)
   ```

   The first part is the user that ran the method, and the numbers in the parentheses are the user ID and group ID actually used to run the method.

   Note: For HP–UX systems, when the numbers in parentheses are less than 0, the method is run as **nobody**.
5. After you gain experience running this test, you can manually run the method by using the **objcall** or **idlcall** commands to reproduce the problem and run tests.

   Note: Typically, you should use the **objcall** and **idlcall** commands only under the guidance of authorized support personnel. If you are going to invoke any methods manually, be sure to review the *Tivoli Management Framework Planning for Deployment Guide* and the *Tivoli Management Framework Reference Manual*.

Debugging the Transaction Manager

Use the **tmcmd** command to force a specified transaction or subtransaction to a particular state. Your support provider might ask you to run the **tmcmd** command to assist in problem diagnosis. The **tmcmd** command is primarily useful when testing and debugging the transaction manager, which is linked inside the object dispatcher. Support might ask you to run this command to recover from a severe or unusual problem.

Attention:: Run this command only with the direction and assistance of Customer Support, as this command can crash your object dispatcher or corrupt your database if used inappropriately.

If you are using Application Development Environment to create your own transactions, use the **tmcmd** command to force a specified transaction or subtransaction to a particular state.

Displaying the Status of Current Transactions and Locks

You can examine current pending transactions, locks, and their states by using the **tmstat** command. Your support provider might ask you to run the **tmstat** command and send its output to assist in diagnosing problems.

Locks are used to ensure data consistency on a given resource. To determine if you have a process lock problem, you can monitor the odb.log file. If this file stays at a constant size or more importantly, consistently grows in size for no obvious reason, you could have a lock problem.

Common Error Descriptions

This section describes common errors that you might encounter when you use the logs and commands described in this chapter. This is not a comprehensive listing of all errors. Refer to your system documentation for more information about system-produced errors.

e=1 This error can arise when system utilities are nonstandard versions. An example of this error is when the built-in **tar** utility is replaced with the shareware **tar** utility (nonstandard version).

e=5 There can be a problem with file read and write permissions. Ensure that the administrator or the account performing a Tivoli action has the correct read and write permissions to the file systems affected.

e=12 This error means that a method exception was thrown. An exception of one method can be caught and handled by a parent method. This error code might or might not be interpreted as an error. If you see this error code, run the **wtrace** command to obtain more information on the exception.

s=6 On HP-UX, this is the same library path problem as the s=9 error code.

s=9 The system library for that command is missing or is corrupted. You might need to use your operating-system-specific tool to investigate what libraries are missing. For example, on Solaris, you use the **ldd** command.

s=10 This error often occurs when a daemon terminates unexpectedly. The error indicates a bus error.

s=11 This error often occurs due to a segmentation violation and is indicated by a daemon terminating unexpectedly.

s=15 A termination signal from the **kill** command was sent. A service was terminated.

Using IDL Commands

The following IDL commands enable you to view and modify object properties.

Attention:: Unless you are very familiar with object attributes or you are experimenting on a properly isolated test system, it is recommended that you always use the **get** option for these commands. The default option for many of these commands is **set**, which can irreparably damage your environment. Recovery from such a situation could include the restoration of a previous database backup, if possible.

Note: For information about object hierarchy, see the *Tivoli Management Framework Planning for Deployment Guide*.

idlattr Gets or sets object implementation attributes

idlcall Performs a Tivoli extended IDL call from the command line

irview Views all method and attribute names and details for a resource

objcall
Performs a Tivoli object call (non-IDL) from the command line

odbls Lists the contents of the Tivoli object database

Note: For the full syntax of these commands, see the *Tivoli Management Framework Reference Manual*.

Object Database Tips

The following sections provide tips on how to work with the object hierarchy.

Listing Contents of an Object Database

When troubleshooting a system with the methods running in a Tivoli environment, you might want to use the **odbls** command to list the contents of the Tivoli object database. This command assists you in learning about your Tivoli region without concern of accidental damage to the database, because it provides an export of the database.

The **odbls** command lists the method attributes, access control lists (ACLs), and roles in paragraph format. It is especially useful for looking up method headers and object attributes. This command is often used in conjunction with the **odstat** output to identify method headers listed in the output.

For Windows NT systems, the object dispatcher service must be stopped to run the **odbls** command. For instructions, see "Stopping and starting the object dispatcher" on page 8.

The following table provides the context and authorization role required for this task.

Activity	Context	Required Role
List contents of an object database	Tivoli region	**super**

A common way to use this tool is to enter the following command, where *file* is the file used to search for specific object IDs or method names:

```
odbls -amilk $DBDIR > file
```

Finding the Method Executable

If a method fails, you might try to find out more about it from the object database. You might also want to find the executable file and verify that it is the correct one. This is easily done if a method works on one system but not another. You can then compare the two. Do one or more of the following:

- Duplicate the problem or action.
- Trace with the **odstat** command and the **wtrace** command.
- Identify the method from the gathered output logs.
- Find the executable file responsible for the method by performing these steps:
  1. To find the behavior object ID, enter the following command:

     ```
     objcall oid resolve method
     ```
  2. To find the type of the method listed in the last line of the result of the command and to find other attributes, such as the default user ID, enter the following command:

     ```
     objcall behavior_object_ID om_stat method
     ```

     The method type can be one of the following:

intrinsic
Method is built into the object dispatcher.

default
The method is looked up for architecture of current system.

platform_type
The method is custom for the specified platform.

3. If you have a default method type, you can find the binary path of the method by entering the following command:
   ```
   objcall behavior_object_ID om_get_definition method default
   ```

The following is an example of how to find the binary of a method:

1. To find the behavior object ID, enter the following command:
   ```
   objcall 1264987995.2.7 resolve avail_space

   1264987995.1.324
   ```
2. To get the type of method, enter the following command:
   ```
   objcall 1264987995.1.324 om_stat avail_space

   CATALOG=
   SET_USER=         !User ID with which the method execute.
   SET_GROUP=        !Group ID with which the method execute.
   EXPORT=TRUE
   EXECUTE=FALSE
   default            !This is the type of the method.
   ```
3. To get the path of the binary of the method, enter the following command:
   ```
   objcall 1264987995.1.324 om_get_definition avail_space default

   STORAGE=/TAS/MANAGED_NODE/man_node_skel1
   MODEL=queued-obj-daemon
   ```

The binary file of the method is at $BINDIR/TAS/MANAGED_NODE (in UNIX) or %BINDIR%\TAS\MANAGED_NODE (in Windows) and the executable file in this example is man_node_skel1.

If the Method Is Unknown

To find all of the methods that run on a particular class object, follow these steps:

1. To find the class object ID (instance manager), enter the following command:
   ```
   wls -od /Library/resource
   ```
2. To get the behavior object ID, enter the following command:
   ```
   idlattr -tg class_object_ID behavior Object
   ```
3. To find all the methods of the object, enter the following command:
   ```
   objcall behavior_object_ID contents
   ```

If you have an instance name for a managed node, follow these steps to find all the methods and their options. This lists all the methods for the class specified, not including those available to an instance through inheritance.

1. To identify the repository ID for a class, enter the following command syntax:
   ```
   wlookup -r ManagedNode instance
   ```
   For example:
   ```
   # wlookup -r ManagedNode spot 1480943038.1.348#TMF_ManagedNode::Managed_Node#
   ```
2. Find the methods and attributes for that class using the *repository_ID*, which is located between the # symbols in the **wlookup** output:

```
irview repository_ID contents
```

For example:

```
# irview TMF_ManagedNode::Managed_Node contents | grep sub
1480943038.1.4##4@TMF_ManagedNode::Managed_Node::update_sub_label
1480943038.1.4##4@TMF_ManagedNode::Managed_Node::add_subscription
1480943038.1.4##4@TMF_ManagedNode::Managed_Node::remove_subscription
1480943038.1.4##4@TMF_ManagedNode::Managed_Node::update_sub_label
1480943038.1.4##4@TMF_ManagedNode::Managed_Node::add_subscription
1480943038.1.4##4@TMF_ManagedNode::Managed_Node::remove_subscription
1480943038.1.4##3@TMF_ManagedNode::Managed_Node::subscriptions
1480943038.1.4##3@TMF_ManagedNode::Managed_Node::subscriptions
```

Getting and setting object attributes using the idlattr command

You can use the **idlattr** command to view attribute values that are not accessible when performing actions through the desktop or with common Tivoli commands. To view attribute values, use the **idlattr** command syntax as follows, where *attrname* specifies the attribute name and *typecode* specifies the fully scoped type of the attribute:

```
idlattr -tg $object_id attrname typecode
```

You also can use the **idlattr** command to set attributes as follows, where *value* specifies the value the attribute is to be set to:

```
idlattr -ts $object_id attrname typecode [value]
```

The challenge to using this command is finding the typecode for the attribute. Commonly, you can find the attribute name from **odstat** or **wtrace** output or by issuing an **objcall** $*object_ID* **contents** command. Next, to find the description for a specific attribute method, enter the following command:

```
irview repository_ID::meth_or_attr describe
```

For example:

```
# irview TMF_ManagedNode::Managed_Node::subscriptions describe
AttributeDescription
 name:          subscriptions
 id:            TMF_ManagedNode::Managed_Node::subscriptions
 defined in:    TMF_CCMS::Subscriber
 TypeCode:      TMF_CCMS::_sequence_ssubscriber_subscriber_list
     kind:       tk_sequence
     to_orb_free: 1
     size:       12
     # parms:     2
 mode:          READONLY
```

Using this information, you can then issue an **idlattr** command. For example:

```
idlattr -tg $managednode_object_ID subscriptions \
   TMF_CCMS::_sequence_ssubscriber_subscriber_list
```

When you use the **idlattr** command to change the value of attributes, this command bypasses normal safeguards that are inherent in similar actions performed through the command line or the desktop.

Invoking methods using the idlcall command

When performing actions using the Tivoli desktop or common Tivoli commands, many method calls are performed for that single action. Sometimes, it is useful to reproduce or work with a single method call when investigating a problem. With information obtained from **odstat**, **wtrace**, and **irview** output, you can reconstruct a single method call and then use the **idlcall** command.

The **idlcall** command syntax is as follows:

```
idlcall $object_id method_name method_args
```

The challenge to using this command is knowing how to provide the method arguments. The object ID and method name are obtained directly from the **odstat** output.

By viewing the **wtrace** output and the **irview** output, you can find the arguments for the method using the following scenario:

1. From the **odstat** output in "odstat and wtrace code examples" on page 76, locate the object ID and method name and reconstruct the **add_subscription** method.
2. Use the **irview** command to find the method call description for a specific method:

   ```
   irview repository_ID::meth_or_attr describe
   ```

   For example:

```
# irview TMF_ManagedNode::Managed_Node::add_subscription \
    describe OperationDescription

name:           add_subscription
id:             TMF_ManagedNode::Managed_Node::add_subscription
defined in:      TMF_CCMS::Subscriber
 TypeCode: void
     kind:        tk_void
     to_orb_free: 0
     size:         0
     # parms:     0
 mode:           NORMAL
    ParameterDescription
    name:       profile_manager
    id:         TMF_ManagedNode::Managed_Node::add_subscription::profile_manager
    defined in:  TMF_CCMS::Subscriber::add_subscription
     TypeCode: TMF_CCMS::ssubscriber

         kind:        tk_struct
         to_orb_free: 1
         size:        40
         # parms:     9
     mode:           IN

    ParameterDescription
     name:          endpoint
     id:            TMF_ManagedNode::Managed_Node::add_subscription::endpoint
     defined in:     TMF_CCMS::Subscriber::add_subscription
     TypeCode: boolean
         kind:        tk_boolean
         to_orb_free: 0
         size:        1
         # parms:     0
     mode:           OUT
     ExceptionDescription
     name:           ExAddSubscriptionPolicyValidation
     id:             TMF_CCMS::ExAddSubscriptionPolicyValidation
     defined in:     TMF_CCMS
     TypeCode: TMF_CCMS::ExAddSubscriptionPolicyValidation
         kind:        tk_struct
         to_orb_free: 1
         size:        36
         # parms:     15
```

3. From this information, you can issue an **idlcall** command. For example, the **idlcall** for this example is as follows:

```
idlcall $managednode_object_ID add_subscription \
'{{{"1480943038.1.580#TMF_CCMS::ProfileManager#" \
 "SwPacks"} "OBJECT_NIL" "" ""} 0 false {0}}'
```

odstat and wtrace code examples

This chapter includes examples of a managed node subscription to a profile manager. The following code examples display the entire output of the **odstat** and **wtrace** commands to help you understand these examples.

Output from the odstat command

Output from the **odstat** command is as follows:

```
337 O+            done   15  0  23:53:56  0.0.0 get_oserv
338 O+            done 2103  0  23:53:58  1480943038.1.2 query odstat
339 O+            done   15  0  23:53:50  0.0.0 get_name_registry
340 O+hdoq        done  846  0  23:53:50  1480943038.1.26 lookup
341 O+hdoq        done  113  0  23:53:50  1480943038.1.26 lookup
342 O+hdoq        done   18  0  23:53:50  1480943038.1.580#TMF_CCMS::ProfileManager# subscribe
343 O+hdq   1-342 done   64  0  23:53:50  1480943038.1.14#TMF_SysAdmin::Library# lookup_object
344 O+hdq   1-342 done    9  0  23:53:50  1480943038.1.577#TMF_PolicyRegion::GUI# is_validation_enabled
345 O+      1-342 done   12  0  23:53:50  1480943038.1.580#TMF_CCMS::ProfileManager# _get_flags
346 O       1-342 done    0  0  23:53:50  <batch-mgr> add_subscription
347 O+ahdoq 1-342 done    9  0  23:53:50  1480943038.1.348#TMF_ManagedNode::Managed_Node# add_subscription
348 O+hdq   1-347 done   64  0  23:53:50  1480943038.1.14#TMF_SysAdmin::Library# lookup_object
349 O+hdq   1-347 done    9  0  23:53:50  1480943038.1.196#TMF_PolicyRegion::GUI# is_validation_enabled
350 O+      1-347 done   40  0  23:53:50  1480943038.1.196#TMF_PolicyRegion::GUI# pm_val_subscription
351 O+hdq   1-342 done  103  0  23:53:50  1480943038.1.348#TMF_ManagedNode::Managed_Node# get_cache_info
352 O+            done   15  0  23:53:50  0.0.0 get_master_base
353 O+            done   15  0  23:53:50  0.0.0 get_oserv
354 O+            done 2103  0  23:53:50  1480943038.1.2 cntl tmstat
355 O+            done   15  0  23:53:59  0.0.0 get_oserv
```

Output from the wtrace command

Output from the **wtrace** command is as follows:

```
loc-ic   339    M-H   Extern        0
      Time run:    [Thu 01-Jun 23:53:50]
      Object ID:    1480943038.1.0
      Method:      get_name_registry
      Method Args: NameRegistry
      Principal:   root@spot (0/0)
      Path:        getattr
loc-oc   339                        15
      Results: (ascii):         1480943038.1.26
loc-ic   340 M-hdoq   Extern       37
      Time run:    [Thu 01-Jun 23:53:50]
      Object ID:    1480943038.1.26
      Method:      lookup
      Principal:   root@spot (60001/60001)
      Path:        /solaris2/TMF/BASESVCS/TNR_prog1
      Input Data: (encoded):
             "ManagedNode"  "spot"
loc-oc   340                      846
      Results: (encoded):
             {
               "1480943038.1.348#TMF_ManagedNode::Managed_Node#" "spot"
               {
                 "TMF_TNR::_sequence_ObjectInfo_ObjectInfoList" 19 false
                {
                   6
                   [
                     {
                    "1480943038.1.349#TMF_ManagedNode::TaskExecute#"
```

```
                      "TaskExecute"
                         {
                           "null" 0 false
                         }
                       }
                       {
                         "1480943038.1.375#TMF_Backup::Client#" "BackupClient"
                         {
                           "null" 0 false
                         }
                       }
                       {
                         "1480943038.1.530#TMF_GWProxy::GWProxy#""GatewayProxy"
                        {
                           "null" 0 false
                         }
                       }
                       {
                         "1480943038.1.545#TMF_UI::DesktopList#" "imp_TMF_UI:
                         :DesktopList"
                         {
                           "null" 0 false
                         }
                       }
                       {
                         "1480943038.1.546#TMF_UI::Extd_DesktopList#" "imp_TMF
                         _UI::Extd_DesktopList"
                         {
                           "null" 0 false
                         }
                       }
                       {
                         "1480943038.1.548#HTTPDaemon::HTTPd#" "HTTPd"
                         {
                           "null" 0 false
                         }
                       }
                     ]
                   }
                 }
               }

loc-ic   341 M-hdoq   Extern       43
      Time run:    [Thu 01-Jun 23:53:50]
      Object ID:   1480943038.1.26
      Method:      lookup
      Principal:   root@spot (60001/60001)
      Path:         /solaris2/TMF/BASESVCS/TNR_prog1
      Input Data: (encoded):
             "ProfileManager"  "SwPacks"
loc-oc   341                      113
      Results: (encoded):
             {
               "1480943038.1.580#TMF_CCMS::ProfileManager#" "SwPacks"
               {
                 "null" 0 false
               }
             }
loc-ic   342 M-hdoq   Extern      173
      Time run:    [Thu 01-Jun 23:53:50]
      Object ID:   1480943038.1.580#TMF_CCMS::ProfileManager#
      Method:      subscribe
      Principal:   root@spot (60001/60001)
      Path:        /solaris2/TAS/CCMS/profile_organizer
      Trans Id:
             {
               1480943038:1,1480943038:1,4:190
```

```
              }
              #4
       Input Data: (encoded):
              {
                1
                [
                  {
                  {
                      {
                        "1480943038.1.348#TMF_ManagedNode::Managed_Node#"
                        "spot"
                      }
                      "OBJECT_NIL" "" ""
                    }
                    0 false
                    {
                      0
                    }
                  }
                ]
              }
loc-ic   343  M-hdq    1-342       37
       Time run:    [Thu 01-Jun 23:53:50]
       Object ID:    1480943038.1.14#TMF_SysAdmin::Library#
       Method:      lookup_object
       Principal:   root@spot (60001/60001)
       Path:        /solaris2/TMF/BASESVCS/Collection_prog1
       Input Data: (encoded):
              "ProfileManager"
              {
                0
              }
loc-oc   343                       64
       Results: (encoded):
              "1480943038.1.288#TMF_SysAdmin::InstanceManager#"
loc-ic   344  M-hdq    1-342       58
       Time run:    [Thu 01-Jun 23:53:50]
       Object ID:   1480943038.1.577#TMF_PolicyRegion::GUI#
       Method:      is_validation_enabled
       Principal:   root@spot (60001/60001)
       Path:         /solaris2/TMF/BASESVCS/Policy_prog1
       Trans Id:
              {
                1480943038:1,1480943038:1,4:190
              },
              {
                1480943038:1,1480943038:1,4:191
              }
              #3
       Input Data: (encoded):
              "1480943038.1.288#TMF_SysAdmin::InstanceManager#"
loc-oc   344                        9
       Results: (encoded):
              false
loc-ic   345    M-H    1-342        0
       Time run:    [Thu 01-Jun 23:53:50]
       Object ID:   1480943038.1.580#TMF_CCMS::ProfileManager#
       Method:      _get_flags
       Method Args: flags
       Principal:   root@spot (0/0)
       Path:        i_getattr
loc-oc   345                       12
       Results: (encoded):
              0
loc-ic   347 M-hdoq    1-342      153
       Time run:    [Thu 01-Jun 23:53:50]
       Object ID:   1480943038.1.348#TMF_ManagedNode::Managed_Node#
```

```
      Method:      add_subscription      Principal:   root@spot (0/60001)
      Path:        /solaris2/TAS/CCMS/profile_organizer
       Trans Id:
             {
               1480943038:1,1480943038:1,4:190
             },
             {
               1480943038:1,1480943038:1,4:192
             }
             #3
      Input Data: (encoded):
             {
               {
                 {
                   "1480943038.1.580#TMF_CCMS::ProfileManager#" "SwPacks"
                 }
                 "OBJECT_NIL" "" ""
               }
               0 false
               {
                 0
               }
             }loc-ic   348  M-hdq    1-347       37
      Time run:    [Thu 01-Jun 23:53:50]
      Object ID:    1480943038.1.14#TMF_SysAdmin::Library#
      Method:      lookup_object
      Principal:   root@spot (60001/60001)
      Path:        /solaris2/TMF/BASESVCS/Collection_prog1
      Input Data: (encoded):
             "ProfileManager"
             {
               0
             }
loc-oc   348                      64
      Results: (encoded):
             "1480943038.1.288#TMF_SysAdmin::InstanceManager#"
loc-ic   349  M-hdq    1-347       58
      Time run:    [Thu 01-Jun 23:53:50]
      Object ID:    1480943038.1.196#TMF_PolicyRegion::GUI#
      Method:      is_validation_enabled
      Principal:   root@spot (60001/60001)
      Path:        /solaris2/TMF/BASESVCS/Policy_prog1
      Trans Id:
             {
               1480943038:1,1480943038:1,4:190
             },
             {
               1480943038:1,1480943038:1,4:192
             },
             {
              1480943038:1,1480943038:1,4:193
             }
             #3
      Input Data: (encoded):
             "1480943038.1.288#TMF_SysAdmin::InstanceManager#"
loc-oc   349                       9
      Results: (encoded):
             true
loc-ic   350    M-H    1-347      105
      Time run:    [Thu 01-Jun 23:53:50]
     Object ID:    1480943038.1.196#TMF_PolicyRegion::GUI#
      Method:      pm_val_subscription
      Method Args: pm_val_subscription
      Principal:   root@spot (0/0)
      Path:        getattr
      Trans Id:
             {
              1480943038:1,1480943038:1,4:190
            },
            {
              1480943038:1,1480943038:1,4:192
            },
            {
              1480943038:1,1480943038:1,4:194
            }
            #3
      Input Data: (encoded):
            {
              3
              [
                "pm_val_subscription" "spot" "SwPacks"
              ]
            }
            {
              0
            }
            {
              0
            }
loc-oc   350                       40
      Results: (encoded):
            {
              4 "0x54 0x52 0x55 0x45 "
            }
            {
              0
            }
              0
loc-oc   347                        9
      Results: (encoded):
            true
loc-ic   351  M-hdq    1-342        0
      Time run:    [Thu 01-Jun 23:53:50]
      Object ID:    1480943038.1.348#TMF_ManagedNode::Managed_Node#
      Method:      get_cache_info
      Principal:   root@spot (60001/60001)
      Path:        /solaris2/TMF/BASESVCS/Collection_prog1
      Trans Id:
            {
              1480943038:1,1480943038:1,4:190
            },
            {
              1480943038:1,1480943038:1,4:195
            }
            #1
loc-oc   351                      103
      Results: (encoded):
            "ManagedNode"  "solaris2-server"  "1480943038.1.347#TMF_UI:
            :Presentation#"
loc-oc   342                       18
      Results: (encoded):
            {
              0
            }
```

Chapter 4. Troubleshooting endpoints

When determining problems with the endpoint, it is important to remember that the endpoint operates in conjunction with two other components of the Tivoli environment: the endpoint manager and the gateway. Therefore, you need to assess the state of several Tivoli Management Framework components when troubleshooting endpoints: the endpoint itself, the gateways it can connect to, and the endpoint manager that is on the Tivoli server.

This chapter contains the following sections:

General troubleshooting procedure for endpoints

It is important to understand the relationship of the endpoint to the gateways and Tivoli server. Problems that manifest themselves on the endpoint can actually be a problem with either a gateway or the Tivoli server.

The gateway provides communication between a group of endpoints and the rest of Tivoli Management Framework. The gateway also maintains a cache of endpoint data, methods, and method dependencies. The Tivoli server contains the endpoint manager, which keeps track of all the endpoints in a Tivoli region. The endpoint manager assists the gateways during logins for new or migrating endpoints. Thus, you should always check to see if there are problems with the Tivoli server and gateways as well as with the endpoint itself.

If an endpoint cannot log in to any gateways, check the following points:

- Did you create a gateway?
- Is the gateway up and is it reachable?
- Is the gateway listening on the port that is expected by the endpoint daemon?
- Did the gateway receive the request?
- Did the gateway forward the login request to the Tivoli server?
- Did the gateway service the request?

Use the **wgateway** command to obtain a list of installed gateway object IDs (OIDs) and labels along with a status, which can be cached data and therefore unreliable. For detailed information about a specific gateway and what port it is using, use the **wgateway** command with the **describe** option. In addition, you can use the **wgateway** command to start the gateway if it is down.

When you restart a gateway, certain conditions can cause the gateway listener to start before the MDist 2 repeater thread. When this occurs, the gateway cannot process any requests and gateway methods will fail. To troubleshoot this problem, check the gatelog. The following example shows a normal gatelog in which the MDist 2 repeater thread started before the gateway listener (tcp_server and udp_server):

```
2004/11/05 17:03:26  8 30067e88: mdist: abdb: pending distributions = 0
2004/11/05 17:03:26  2 30067e88: mdist: repeater thread started...
2004/11/05 17:03:26  8 30067e88: mdist: repeater waiting on any event.
2004/11/05 17:03:34  5 30020b68: boot: Starting gateway listener
2004/11/05 17:03:35  5 30020b68: boot: gateway boot completed successfully.
2004/11/05 17:03:35  3 300dbc08: tcp_server: listening on 0.0.0.0+4949
2004/11/05 17:03:35  3 300dc2a8: udp_server: listening on 0.0.0.0+4949
```

The following gatelog shows the condition in which the gateway listener started before the MDist 2 repeater thread:

```
2004/11/05 17:03:26  8 30067e88: mdist: abdb: pending distributions = 0
2004/11/05 17:03:34  5 30020b68: boot: Starting gateway listener
2004/11/05 17:03:35  5 30020b68: boot: gateway boot completed successfully.
2004/11/05 17:03:35  3 300dbc08: tcp_server: listening on 0.0.0.0+4949
2004/11/05 17:03:35  3 300dc2a8: udp_server: listening on 0.0.0.0+4949
```

When this condition occurs, use the **wgateway** *gateway_label* **start_delay_gwlistener** command to delay the start of the gateway.

For more information about the **wgateway** command, see *Tivoli Management Framework Reference Manual*.

When you have problems with endpoint communication or login, run the following commands gather information. The *ep_label* string identifies the endpoint on which the other **wep** suboptions are run:

wep *ep_label* **get address**
: Returns IP address information from the endpoint manager about an endpoint. This command is useful when you need to find the IP address of a specified endpoint.

wep *ep_label* **get gateway**
: Returns gateway information from the endpoint manager about an endpoint. This command is useful when you need to find out the gateway of a specified endpoint.

wep *ep_label* **status**
: Provides status information on a single endpoint basis. If you receive the message, `*ep_label* is alive`, you can run a method on an endpoint. Communication is successful between the endpoint manager, gateway, and endpoint. Otherwise, this command returns the `unable to determine endpoint status; endpoint may be unreachable` error message. This message indicates that there is a communication failure to the endpoint. This failure might be between the endpoint manager and the gateway or between the gateway and the endpoint.

wepstatus *ep_label...*
: Provides status information about the specified endpoint or endpoints. The endpoint status can be one of the following values:

  connected
  : The endpoint is connected to a gateway. The endpoint and gateway can communicate. The endpoint responds to downcalls from the gateway, can run a task, and can initiate an upcall. The gateway services the upcall initiated by the endpoint.

disconnected
: The gateway knows that the endpoint has disconnected from the gateway and does not try to communicate. The endpoint is logged out of the gateway, but might still be connected to the network.

unavailable
: The gateway cannot reach the endpoint for one of the following reasons:
  - The gateway cannot make a successful downcall.
  - The endpoint cannot run a task.
  - The endpoint is not servicing upcalls.
  - The endpoint is out of disk space.
  - There is a permission problem on the endpoint that is preventing the endpoint from reading or writing to a file or running a program.

unreachable
: The gateway cannot reach the endpoint. The endpoint process might have been stopped or the endpoint might be disconnected from the network.

unknown
: The gateway associated with the endpoint is down. or the gateway has not checked the status of the endpoint. The wepstatus command is unable to determine the status of the endpoint.

wep ls
: Returns information about endpoints from the endpoint manager. This command is useful when you need to find out which endpoints are assigned to each gateway. You might find that an endpoint is assigned to a gateway other than the one expected.

wepmgr
: Provides control and configuration for the endpoint manager. You can start, stop, and restart the endpoint manager. In addition, this command gets and sets endpoint manager attributes in the Tivoli object database to control endpoint login.

wgateway
: Starts, stops, or lists the properties of an endpoint gateway. This command also provides a list of installed gateway object IDs, labels, and status. If you want to know if the gateway is operating properly and what port it is using, use the **wgateway** command with the **describe** option.

wlookup –ar Endpoint
: Returns information stored in the name registry about the endpoint on the Tivoli server. Compare the information returned from the **wlookup** command to the information returned by the **wep** command with the **ls** option. There can be inconsistencies between the endpoint manager and the Tivoli name registry. To correct any inconsistencies, use the **wepmgr** command with the **fsck** option to rewrite the data in the name registry with the data from the endpoint manager. For more information about the **wepmgr** command, see "Mismatches between the endpoint manager and the name registry" on page 97.

If you cannot detect a problem with these methods, the next step is to check the log files. The following section describes how to use debugging levels and log files to determine endpoint problems.

Using log files to troubleshoot endpoints

View the epmgrlog, gatelog, and lcfd.log files to determine and investigate problems with endpoints. Keep in mind that unlike the **lcfd.log** file, which is re-created each time the endpoint process starts, the epmgrlog and gatelog files grow indefinitely and must be truncated or archived on a regular basis. For more information, see "Using Log Files" on page 59.

The epmgrlog file, found on the Tivoli server, provides information about the operations of the endpoint manager. This file is located in the database directory. View this log to discover if the error is occurring in the endpoint manager. By examining this file, you can review messages concerning endpoint login and migration from the standpoint of the endpoint manager.

The **gatelog** file, found in the database directory on the managed node where the gateway is installed, contains information about the behavior of the gateway. You can change the debugging level for this log with the **wgateway** command with the **set_debug_level** option. The recommended debugging level is 6.

The lcfd.log file, found on each endpoint in the lcf/dat directory, contains logging messages for upcall methods, downcall methods, and the login activities of the endpoint. You also can view this log file from the http interface. In addition, lcfd.log can have different levels of debugging information written to it. To set the level of debugging, use the **lcfd** command with the **–d***level* option, which sets the **log_threshold** option in the last.cfg file. Set the **log_threshold** at level 2 for problem determination, because level 3 often provides too much information.

Of the three log files, the lcfd.log file is sometimes the most useful for debugging endpoint problems. However, remote access to the endpoint is necessary for one-to-one contact.

Endpoint log messages have the following format:

```
timestamp level app_name message
```

The message elements are as follows:

timestamp
: Displays the date and time that the message was logged.

level
: Displays the logging level of the message.

app_name
: Displays the name of the application that generated the message.

message
: Displays the full message text. The content of *message* is provided by the application specified in *app_name*.

The default limit of the log file is 1 megabyte, which you can adjust with the **lcfd** (or **lcfd.sh**) command with the **–D log_size** =*max_size* option. The valid range is 10240 through 10240000 bytes. When the maximum size is reached, the file reduces to a size of approximately 200 messages and continues to log. "Using Log Files" on page 59 discusses the epmgrlog, the gatelog, and the lcfd.log files in more detail.

In addition to these three log files, the following files help troubleshoot endpoint problems located on the endpoint:

last.cfg
A text file that contains the endpoint and gateway login configuration information from the last time the endpoint successfully logged in to its assigned gateway. Use this file to review the configuration settings for an endpoint.

lcf.id A text file that contains a unique ID number to represent the endpoint. This file is uniquely generated if the TMEID.tag file does not exist.

lcf.dat A binary file that contains the gateway login information. You cannot modify this information; however, you can view network configuration information from the http interface.

Of these files, the last.cfg file can be useful in determining problems with an endpoint. The last.cfg file resides in the \dat subdirectory of the endpoint installation and also can be viewed from the http interface. This file contains configuration information for the endpoint. The following example shows the contents of a last.cfg file:

```
lcfd_port=9495
lcfd_preferred_port=9495
gateway_port=9494
protocol=TCPIP
log_threshold=1
start_timeout=120
run_timeout=120
lcfd_version=41100
logfile=C:\Program Files\Tivoli\lcf\dat\1\lcfd.log
config_path=C:\Program Files\Tivoli\lcf\dat\1\last.cfg
run_dir=C:\Program Files\Tivoli\lcf\dat\1
load_dir=C:\Program Files\Tivoli\lcf\bin\w32-ix86\mrt
lib_dir=C:\Program Files\Tivoli\lcf\bin\w32-ix86\mrt
cache_loc=C:\Program Files\Tivoli\lcf\dat\1\cache
cache_index=C:\Program Files\Tivoli\lcf\dat\1\cache\Index.v5
cache_limit=20480000
log_queue_size=1024
log_size=1024000
udp_interval=300
udp_attempts=6
login_interval=1800
lcs.machine_name=andrew1
lcs.crypt_mode=196608
lcfd_alternate_port=9496
recvDataTimeout=2
recvDataNumAttempts=10
recvDataQMaxNum=50
login_timeout=300
login_attempts=3
```

For more information about most of these attributes, see the **lcfd** command in the *Tivoli Management Framework Reference Manual*.

When you change endpoint configuration with the **lcfd** command, the last.cfg file changes. Therefore, you should not modify the last.cfg file. If you require changes, use the **lcfd** command to make any changes. However, running the **lcfd** command requires stopping and restarting the endpoint.

Another useful tool for endpoint problem determination is the output from the **wtrace** command. The **wtrace** command is useful for tracking upcall and downcall method failures. To learn more about the **wtrace** command, see Chapter 3, “Troubleshooting the Tivoli environment,” on page 57.

Using Web reporting to troubleshoot endpoints

Tivoli Management Framework enables you to post endpoint status to a Web server. When this feature is enabled, endpoints post status information to a Web server when they boot up and when certain events or state changes occur. The primary advantage of this feature is that endpoints that are not communicating with the Tivoli environment can post status information to a single location, which enables you to quickly locate and troubleshoot the endpoints.

Endpoint Web reporting is disabled by default. To enable and use endpoint Web reporting, you must perform the following tasks:

- Install a Web server.
- Set the **web_post_interval** and **web_post_url** options using the **lcfd –D** or **wep set_config** command. The **web_post_interval** option specifies the interval at which the endpoint reports events and state changes. The **web_post_url** option specifies the URL for the Web post. For more information about these commands and options, see the *Tivoli Management Framework Reference Manual*.
- Create a CGI script to process the endpoint posts. Endpoints post information to the Web server as standard input. Tivoli Management Framework provides a sample Perl script named epwebpost_handler.pl that processes this input and writes it to a log file named epwebpost.log. You can use this script as it is, edit the script to suit your needs, or create your own script. For example, you could create a script that loads endpoint post data into a database or sends Tivoli Enterprise Console events.

  The epwebpost_handler.pl script is located on the Tivoli Management Framework 2 of 2 CD in the LCF\WEBPOST directory.

When endpoint Web reporting is enabled, endpoints post status information to a Web server when they boot up. Then, after the interval set by the **web_post_interval** parameter, the endpoint checks to see if an event or state change has occurred. If an event or state change has occurred, the endpoint posts the information to the Web server.

The endpoint posts the following information to the Web server:

time_stamp
: The time when the packet of endpoint status information is sent to the Web server.

last_restart
: The time, in hh:mm:ss format, that the endpoint was last started.

up_time
: The time, in days, hh:mm:ss format, that the endpoint has been running.

run_state
: The current run state of the endpoint. Possible states are `login`, `running`, `standby`, and `shutdown`.

interp
: The interpreter type of the endpoint.

lcfd_version
: The revision level of the endpoint.

machine_unique_id
: The unique identifier for the installed endpoint.

machine_name
: The endpoint label.

hostname
The hostname for the endpoint.

local_ip_interface
The IP address configured for the endpoint to bind by the local_ip_interface parameter in the last.cfg configuration file. If no configuration is performed, the value is `0.0.0.0` (`ANY_ADR`).

lcfd_port
The current bound listening port for the endpoint. This value might be different from the preferred and alternate port fields.

lcfd_preferred_port
The preferred port for the endpoint.

lcfd_alternate_port
The alternate port for the endpoint.

lcfd_mac_addr
The MAC address of the endpoint.

gateway_ip_addr
The gateway IP address that the endpoint is currently logged in to.

gateway_port
The current configured gateway port number.

win_os_platform
For Windows systems, the operating system that is installed on the system. This parameter can have one of the following values:

WinNT
Windows NT, Windows 2000, Windows XP, or Windows Server 2003

Win32 Windows 95, Windows 98, or Windows Me

win_os_type
For Windows systems, whether the operating system is a server or desktop operating system.

win_os_ver
For Windows systems, the version of the operating system.

login_interfaces
The list of gateway IP addresses that are available for the endpoint to contact during login operations.

events The list of events or state changes that have occurred during the monitoring interval configured by the **web_post_interval** parameter.

Note: The Web post ″events=″ variable contains a colon-delimited list of integers. The event codes are generated either by state transitions (login, logout, and so forth.) or error since the last Web post.

Description of event codes:
01 - STATECHANGE: endpoint process on this system has been SHUTDOWN.
02 - STATECHANGE: endpoint process on this system entered STANDBY mode.
03 - STATECHANGE: endpoint process on this system has RESUMED operation.
04 - STATECHANGE: endpoint successfully started its IPC server.
05 - STATECHANGE: endpoint shut down its IPC server.
06 - STATECHANGE: endpoint detected a change in the local network

configuration.
07 - STATECHANGE: endpoint process on this system has been STARTED.
10 - LOGIN: endpoint has successfully login to a gateway and is running OK.
11 - LOGIN: endpoint login to gateway attempt failed. The gateway is unreachable, or down.
12 - LOGIN: endpoint is migrating to a new gateway as the result of a migration.
13 - LOGIN: endpoint has been redirected to another Tivoli region.
20 - UPCALL: endpoint upcall to gateway failed because the gateway is unreachable or down.
21 - UPCALL: Warning, method execution was performed without a valid user ID.
30 - RUNIMPL: Method failed because the endpoint was unable to find or run the method implementation.
31 - RUNIMPL: Method failed to complete because the method threw an exception before execution could complete.
35 - RUNTEST: Unable to generate a token for the built-in Administrator.
36 - RUNTEST: Unable to generate a token for the tmersrvd account.
37 - RUNTEST: Unable to spawn test process for endpoint self-check.
40 - RUNTASK: No appropriate executable is defined for this INTERP.
41 - RUNTASK: Failed to create task file on endpoint.
42 - RUNTASK: Failed to set required permission on temporary file.
43 - RUNTASK: Error configuring required I/O streams.
44 - RUNTASK: Failed to get user token for task.
45 - RUNTASK: Failed to execute task.
50 - TAPCALL: Tivoli Authentication Package initialization failed.
51 - TAPCALL: Failed to initialize Tivoli Authentication Package token pool.
52 - TAPCALL: Tivoli Authentication Package token cache has expired.
53 - TAPCALL: Tivoli Authentication Package Account Name lookup failed.
60 - HMAC: Endpoint received data that could not be decrypted.
65 - RUNCMD: Error spawning intrinsic command.
66 - RUNCMD: Error redirecting input or output for intrinsic command.
67 - RUNCMD: Error waiting for intrinsic command to terminate.
68 - RUNCMD: Error retrieving exit code from intrinsic command.
70 - REPAIR: Error creating or writing file(s) in TEMP directory.
71 - REPAIR: Requested amount of TEMP space is not available.
72 - REPAIR: Error creating or writing file(s) in CACHE directory.
73 - REPAIR: Requested amount of CACHE space is not available.
74 - REPAIR: The Tivoli Authentication Package is not installed.
75 - REPAIR: A reboot is required to activate the Tivoli Authentication Package.
76 - REPAIR: The tmersrvd account does not exist (or is misconfigured).
77 - REPAIR: The Tivoli_Admin_Privileges group does not exist (or is misconfigured).
78 - REPAIR: The tmersrvd account does not permission to read files in System32.
79 - REPAIR: The lcfd service is not set to autostart.

The following example shows data contained in the epwebpost.log file for a UNIX system:

```
================[ENDPOINT HEALTH]====================
Content-length:433
Remote-addr:9.48.135.135
```

```
Remote-port:1247
time_stamp=2003 09 23 11:15:03
last_restart=2003 09 23 11:10:02
up_time=0 days, 00:05:01 h:m:s
run_state=Running
interp=w32-ix86
lcfd_version=41100
machine_unique_id=QWDP0G6 26G0G9WCYCZQ00000554
machine_name=meerkat
hostname=meerkat
local_ip_interface=0.0.0.0
lcfd_port=3188
lcfd_preferred_port=3188
lcfd_alternate_port=9496
lcfd_mac_addr=00-00-76-37-E5-10
gateway_ip_addr=9.44.125.8+2525
gateway_port=2525
login_interfaces=9.44.125.8+2525
events=52:10
TAPCALL: Tivoli Authentication Package token cache has expired.
LOGIN: LCFD has successfully login to a gateway and is running OK.
```

Common difficulties with endpoint login

Consider the following questions when endpoint logins are unsuccessful:

- Did the endpoint log in as a new endpoint in the Tivoli region?
- Did the endpoint log in as part of the normal login sequence?
- Did the endpoint migrate to a different gateway?

This section provides some insight on what factors could be keeping the login process from working properly.

Note: Before attempting to install endpoints and have them log in, it is important to review the *Tivoli Management Framework Planning for Deployment Guide* to properly plan and implement your particular strategy for endpoint logins.

If you are having problems with endpoint login, review the following checklist to ensure that you have properly implemented the login process for your Tivoli environment:

- Are you using broadcast as the means for endpoint login? If so, the network, gateway, and the endpoint manager could be flooded with more login requests than they can accommodate. You might want to employ one of these alternate login strategies:
  - Use **allow_install_policy**, **after_install_policy**, **login_policy**, and **select_gateway_policy** policy scripts to better manage the performance of your gateways and endpoint manager. For more information about endpoint policy, see the *Tivoli Management Framework Reference Manual*.
  - Use the endpoint manager attributes of the **wepmgr** command, **max_install**, **max_sgp**, and **max_after**, to throttle the number of concurrent endpoint policy scripts, and the **login_interval** attribute to control how often an individual endpoint can attempt to log in. For more information about the **wepmgr** command, see *Tivoli Management Framework Planning for Deployment Guide*. The **login_interval** attribute on the endpoint manager is different than the **login_interval** attribute in the last.cfg file on the endpoint.

  Another problem with broadcast logins occurs when the endpoint logs in to multiple gateways. Because of the **login_interval** attribute on the server, duplicate login attempts are recognized and filtered. However, if the gateways are in separate Tivoli regions, the last login request is honored, and the rest return error messages on downcalls and upcalls.

- If you are not using broadcast for endpoint logins, determine which gateways are being used by looking at the **lcs.login_interfaces** setting in the last.cfg file.
- Determine if the endpoint manager **login_interval** attribute value conflicts with the other attributes or with the endpoint **udp_interval**, **udp_attempts**, **login_timeout**, **login_attempts**, or **login_interval** values (set by the **lcfd** command). The endpoint manager **login_interval** attribute controls the amount of time between when an endpoint can log in to a single gateway or perform an initial or isolation login to the endpoint manager. If the **udp_interval** or **login_timeout** for the endpoint is less than the endpoint manager **login_interval**, it can cause the endpoint to become locked out until the endpoint **login_interval** time is reached.
- Determine if the gateways are up in the Tivoli region. Use the **wgateway** command to find the status of a gateway and to restart it, if necessary.
- Determine if the Tivoli server is reachable. The Tivoli server must be available on the network during endpoint login. The endpoint login relies on the gateway and the Tivoli server. Use the **wping** command to determine if the Tivoli server can be contacted. If the object dispatcher responds, this command prints the following to standard output:

  ```
  object dispatcher on host_name is alive
  ```
- If you are using TCP/IP, determine if the network routers are configured to forward UDP broadcasts. If so, an endpoint could log in to multiple gateways using broadcast login.
- If you are using IPX/SPX and you try to login an endpoint using the gateway server name or the extended broadcast login, verify that the Novell NetWare RIP configuration is correct and that the requested gateway is within a 5 hops radius

The following sections provide scenarios of problems that can occur during endpoint initial login and as a result of the login.

When an endpoint login takes too long or does not complete

If a gateway and an endpoint are installed on the same system, it can sometimes take up to 10 minutes for the endpoint to log in to the gateway. The reason for this delay is because the lcfd endpoint process must wait until the gateway finishes its startup procedure. If the endpoint attempts to log in before the assigned gateway on the same system is up, the endpoint login to the assigned gateway fails, and the endpoint attempts an isolation login through its interface list.

To prevent the endpoint from attempting to log in to the gateway before the gateway is up, use the **lcfd** command with the **–D start_delay** option. This option enables you to set a specific number of seconds to force a delay in the login of the endpoint to the gateway. This value varies greatly from installation to installation, depending on the number of endpoints installed to a particular gateway. The larger the number of endpoints, the longer a gateway takes to become fully operational.

To determine which value to use for the **start_delay** option, first view beginning and ending markers in the $DBDIR/gatelog file for the gateway boot process. Next, calculate the time difference between the start and stop time, and add a 5 to 10 minute buffer to the value. For example, suppose that the beginning marker in the gatelog file is:

```
2000/06/19 14:36:00 +06: gateway boot: started booting.
```

Suppose the ending marker is:

```
2000/06/19 14:54:00 +06: gateway boot: gateway boot completed successfully.
```

To determine the optimal value for the **start_delay** option, calculate the difference between the markers (18 minutes) and then add 5 minutes to the value, for a value of 23 minutes.

Creating duplicate endpoints

Duplicate endpoints indicate that one endpoint code installation exists on a host, but multiple entries for that host exist in the gateway database on the server. For example, *ep_label* in the following **wep ls** output displays four entries, with all but one entry displaying a period (.) and a dispatcher number appended to the host name:

```
ep_label
ep_label.2145
ep_label.2167
ep_label.2234
```

In past releases, if the label was already in use at initial login, the dispatcher number was automatically appended to the endpoint label to create a unique label. This resulted in multiple entries for the same endpoint label, with only one of these entries being the functional endpoint.

In this release, duplicate endpoints occur only when you choose to configure the **allow_install_policy** policy script to exit with code 10. Possible ways to configure this script include the following:

- Exit with code 0. This code enables only initial logins from endpoints with unique labels to succeed. Initial logins fail and an error message is returned if the endpoint label is a duplicate.
- Exit with code 10. This code enables duplicate endpoints to succeed during initial login. A dispatcher number is appended to the end of the endpoint label; thus, creating a unique label.
- Exit with code 0 or code 10. This depends on what you set the environmental variables to when an initial login is taking place with a duplicate endpoint.

The **allow_install_policy** script is set with the following environmental variables so that you can examine the script and decide whether or not to allow this endpoint in the Tivoli region:

LCF_DUPL_OBJECT
: Specifies the object ID of the existing endpoint

LCF_DUPL_ADDRESS
: Specifies the network address of the existing endpoint

LCF_DUPL_GATEWAY
: Specifies the object ID of the existing endpoint gateway

LCF_DUPL_INV_ID
: Specifies the inventory ID of the existing endpoint

LCF_DUPL_INTERP
: Specifies the interpreter type (operating system, platform, or machine type) of the existing endpoint

Causes of duplicate endpoints include the following:

- Installing a machine and reusing the same host name or IP address

- Reinstalling an endpoint on a machine without removing the endpoint from the endpoint manager database
- A congested endpoint manager does not respond to an initial login request before an endpoint times out and submits another initial login request
- The endpoint manager completes an initial login request, but the information fails to be received by the endpoint

In these situations, you must examine the cause and take corrective action or change the **allow_install_policy** script to remedy the situation and create the desired outcome. Create either a shell or Perl script, and test the policy scripts thoroughly using simulated login strings. You also should ensure that your script behaves appropriately for both existing (and non-existent) endpoint host names.

Tivoli Management Framework prevents any further initial logins from creating multiple entries for the same endpoint label. However, if duplicate endpoints exist from previous installations, follow these steps to delete them:

1. To determine if duplicate endpoints exist, use the **wep ls** command.
2. For each duplicate endpoint, enter the following command to determine its status, where *ep_label.xxxx* is the duplicate endpoint label:
   ```
   wep ep_label.xxxx status
   ```
3. Do one of the following:
   - If **wep** returns an unreachable status, use the **wdelep** command to delete the duplicate entry from the endpoint list.
   - If **wep** returns an unreachable message for each of the duplicate endpoint entries, use the **wdelep** command to delete all the duplicate endpoint entries. Next, shut down the lcfd process, delete the lcfd.dat file on the endpoint, and then restart the lcfd process.
   - If **wep** returns a message stating that the endpoint is alive, this indicates that this endpoint is the functional endpoint. Check to see if the endpoint label has a dispatcher number appended to its label. If so, rename the endpoint using the **wep** command with the **set_label** option.

   Note: If orphaned endpoints are enabled (see "Unexpected results with endpoint migration" on page 97), you can delete the duplicate endpoint entries and then simply wait for the endpoint to attempt an isolated login. The endpoint manager recognizes the endpoint as an orphaned endpoint and re-adds it to the endpoint manager.

Duplicate entries for the same endpoint also occur in the case where there are multiple gateways from different Tivoli regions, which are picking up a broadcast login request. In this case, there is an endpoint entry in two different Tivoli regions, but only one of the regions is able to communicate with the endpoint.

To correct this situation, use the **wdelep** command to delete all the duplicate endpoint entries. Next, shut down the lcfd process, delete the lcfd.dat file on the endpoint, and then restart the lcfd process. Use directed login of new endpoints to specified gateways and disable broadcast logins during initial deployment. You also should ensure that only one gateway on the same subnet is reachable by broadcast for all prospective endpoints.

Endpoint manager is unavailable during initial login

In some cases, the endpoint manager can be unavailable during the initial login of an endpoint. This could occur due to the object dispatcher going down, the network connectivity to the Tivoli server failing, or the endpoint manager being overloaded with numerous endpoint logins. The Tivoli server also might become unavailable to endpoint logins if it is busy with a large distribution. It is important to note that the endpoint manager is a component of the Tivoli server and is affected by any outside impacts to the Tivoli region. Thus, if the computer on which the Tivoli server resides becomes unavailable, the Tivoli server is not able to service the initial login requests from endpoints.

Several mechanisms govern initial login. These mechanisms allow you to stagger or throttle a large amount of initial logins coming to the endpoint manager. The endpoint continues to attempt to log in indefinitely based on the **login_interval** cycle set using the **lcfd** command. This interval allows you to set a wait state for the endpoint to attempt its next login based on the details of when you expect your endpoint manager to become available again. (The default setting is 1800 seconds or 30 minutes.) Furthermore, you could have computers with the endpoint daemon preloaded before you install and deploy the Tivoli server and gateways. In any case, the endpoint eventually performs its initial login whether or not the endpoint manager is available during its first login attempt.

To determine why the endpoint manager is busy, view the epmgrlog file. Next, attempt to resolve the load problem. To reduce the impact of an endpoint login to the Tivoli server, you can reset the **login_interval** to shorten the waiting period before an endpoint executes another login attempt. In addition, you might need to tune the computer on which the Tivoli server resides to better handle large amounts of processes running at one time. See “Improving performance in a Tivoli environment” on page 32 for information about tuning suggestions for a Tivoli server.

Gateway is unavailable at initial login

If a gateway is unavailable during initial login, the endpoint could still log in successfully. However, the outcome of this process can reveal some unexpected results. If the specified gateway is not available, the endpoint attempts to log in to an alternate gateway. If you are not expecting the endpoint to log in to a different gateway, this can cause the endpoint to be assigned to an unexpected gateway.

During initial login, at least one gateway must be available to receive login requests. You can specify this gateway in one of the following ways:

- From a list of gateways returned by the **select_gateway_policy** script
- By using the **lcfd** command with the **–g** option
- By specifying additional gateways using the **login_interfaces** attribute of the **lcfd** command or the InstallShield installation program or **winstlcf** commands
- By having the endpoint broadcast for login

If you specify a list of gateways for the endpoint to contact and the gateway you expect the endpoint to log in to becomes unavailable, you should continue to wait for the endpoint to complete its login cycle. The endpoint goes through its login options, such as **login_attempts** and **login_timeout**, before abandoning the login attempt to the desired gateway. After that timeout interval (the default is 15 minutes, or three attempts at five–minute intervals), the endpoint then falls back on the list of additional gateways, if specified, and then broadcasts. Broadcast login is enabled for endpoint login by default. For a list of gateways, you must manually specify the gateways in which you want the endpoint to attempt to log in.

Another initial login scenario is when the endpoint is assigned to the intercepting gateway instead of its specified gateways because the gateways are down. If the specified gateways are down, the intercepting gateway will become the primary gateway for the endpoint, and it is the only gateway that the endpoint will try to log in to as long as that gateway remains active. In this scenario, the intercepting gateway forwards the initial login request of the endpoint to the endpoint manager, which then redirects the login request to the specified gateways. When the connection fails between the endpoint manager and the specified gateways, the endpoint manager assigns the endpoint to the intercepting gateway. Upon the next login attempt by the endpoint, the endpoint again goes through its list of gateways to log in to, and if any of those gateways are available at that time, the endpoint logs in to the gateways specified by the **select_gateway_policy** script.

For more information about endpoint logins and how to configure login parameters, see *Tivoli Management Framework Planning for Deployment Guide*.

Endpoint manager is unavailable at normal login

Unlike initial login, the normal login for an endpoint behaves slightly different in regards to the endpoint manager. If the endpoint manager is unavailable, the endpoint is still able to log in to a gateway. The Tivoli server, however, must be available where the endpoint manager process is running. This is because the gateway runs a login policy method, which must be authenticated by the object dispatcher on the Tivoli server.

The boot methods for the endpoint do not run because the endpoint manager is unavailable. Boot methods for the endpoint require access to the endpoint manager.

To ensure that the endpoint manager is available, use the **wepmgr** command with the **ping** option. If the endpoint manager is down, use the **wepmgr** command with the **start** option to restart it. Then, to restart the endpoint so that it begins normal login, issue the **lcfd** command. It is important for the boot methods to run on normal login.

Gateway is unavailable during normal login

If a gateway is unavailable during normal login for an endpoint, the endpoint can still successfully log in by a process called isolation login. When an endpoint cannot contact its assigned gateway, the endpoint attempts to log in to any gateways specified in its lcf.dat file. If those gateways are also unavailable, the endpoint indefinitely repeats the login cycle every **login_interval** seconds (set by the **lcfd** command).

For example, during testing and troubleshooting of your Tivoli environment, you might remove a gateway from a computer. If the IP address and host name changes, this can cause some problems for endpoints that are assigned to that gateway. For example, if an endpoint is a mobile computer, such as a laptop computer, the endpoint might not have been informed of its new gateway assignment because it has not yet been connected to the network.

This problem arises because the gateway assignment is recorded in the lcf.dat file on the endpoint, and the gateway is identified by the IP address or host name. When the mobile computer returns to the network, the new gateway might not have been assigned the endpoint. If you do not take action to assign that endpoint to another gateway, the endpoint will try unsuccessfully to contact the old gateway based on the gateway IP address or host name. Moreover, distributions to the endpoint might fail to the endpoint until the situation is corrected.

There are two ways to correct this situation. Either you can wait for Tivoli Management Framework to correct the situation automatically or you can issue a series of **wep** commands to remedy the situation.

In the first case, the endpoint attempts to find a new gateway when it restarts or it sends an upcall. This scenario will only be successful if broadcast login is enabled on the endpoint or there are other available gateways listed in the **login_interfaces** list for the endpoint. If you want to specify a particular gateway, which does not appear in the **login_interfaces** list, then you will modify the **select_gateway_policy** script to add the new gateway.

In the second case, you can still migrate the endpoint by performing the following procedure:

1. To migrate the endpoint to the new gateway, enter the following command:

   ```
   wep ep_label migrate -f gw_label
   ```

   No endpoint action is required when using this command. This command modifies the gateway that the endpoint is assigned to in the endpoint manager. Using the **–f** option forces the migration even if the old gateway is unavailable.
2. To send the new gateway assignment to the endpoint, enter the following command:

   ```
   wep set gateway -e ep_label
   ```

   This command prompts the newly-assigned gateway from step 1 to contact the endpoint and inform it of its new gateway assignment. Using the **–g** *gw_label* option (instead of the **–e** option) prompts the gateway to send the gateway assignment to all the endpoints assigned to the gateway specified by *gw_label*.
3. To provide the endpoint with a new list of gateways it can log in to, enter the following command, where *gw_name* is the name of the machine the gateway is on:

   ```
   wep set interfaces -e ep_label gw_name+port
   ```

   Use this command to supply the **login_interfaces** list with new gateways. Using the **–e** option sets the list of gateways for a specified endpoint. You can use the **–g** option if you want to set the list of gateways for all the endpoints assigned to the gateway.

To avoid this situation in the future, you should attempt to migrate all endpoints to another gateway before removing a gateway from a Tivoli region.

Endpoint rescue

Sometimes an endpoint can fail to log in, even after trying all the gateways in its **login_interfaces** list and resorting to the broadcast login. To rescue the endpoint when it is unable to log in, try to modify the endpoint login interval with the **lcfd** command with the **login_interval** attribute. The login interval provides a mechanism where the endpoint is in a wait state until the network problems or gateway accessibility problems disappear. This interval is the amount of time before the next login attempt after the endpoint fails to log in (gateways in its **login_interfaces** list did not respond and the broadcast login fails). By default, this interval is set to 30 minutes (1,800 seconds). You can adjust the login interval if the default interval is not an appropriate setting for your Tivoli environment and network topology. At times, you might want a longer interval for logins, such as the following:

- The intercepting gateway has a known down period or has not yet been created.
- There are transient routing problems.

You also can use the **wep** command to reach the endpoint trying to log in; however, this is only successful after the endpoint has successfully performed its initial login. While the endpoint is pausing during the login interval, it is listening for input. During this time, you can specify what gateway to use or a new set of login interfaces by using the **wep** command with the **set gateway** and **set interfaces** options.

You can use the following actions together or independently to address this situation:

- Modify the login interval of the endpoint with the **lcfd** command.
- Update the gateway information for the endpoint with the **wep** command with the **set gateway** and **set interfaces** options.

Common difficulties after endpoint login

After endpoints have logged in to their gateways, some can interrupt normal management operations on endpoints. The following sections describe some of the more common situations you can encounter.

Orphaned endpoints

An endpoint is *orphaned* when the Tivoli server identifies and deploys an endpoint, but the endpoint is no longer recorded in the endpoint manager and name registry.

Orphaned endpoints occur because of the following situations:

- An endpoint is created during the time frame between when a Tivoli object database is backed up and later restored
- An endpoint is deleted with the **wdelep** command

Normal login fails for orphaned endpoints, meaning that no record of the endpoint exists on the gateway. When the endpoint attempts an isolated login, the endpoint manager recognizes the endpoint as an orphaned endpoint and re-adds it to the endpoint manager.

Ways the Tivoli server determines if an endpoint is orphaned include the following:

- The endpoint has an object ID with a region number, which matches the Tivoli server, and the endpoint dispatcher number is *not* in the endpoint database.
- The endpoint has an object ID with a region number, which matches the Tivoli server, and the endpoint dispatcher number is in the endpoint database. Also, its private key does not match.

  Note: A private key is assigned to the endpoint during initial login and is saved in the endpoint record information.

To register the endpoint with the endpoint manager, follow these steps:

1. To enable orphaned endpoints, enter the following command to set the endpoint manager flag:
   ```
   wepmgr set epmgr_flags 1
   ```

Note: Enable orphaned endpoints only when restoring from a backup or inadvertently deleting an endpoint.

2. To refresh the attributes in the running endpoint manager from the attributes on the endpoint manager object in the Tivoli object database, enter the following command:

   ```
   wepmgr update
   ```

A new endpoint record is created, which includes a new dispatcher number and secret key. This information is passed back to the endpoint. To disable orphaned endpoints, reset the **epmgr_flags** attribute to `0` with the **wepmgr** command. For example, enter the following command:

```
wepmgr set epmgr_flags 0
```

Unexpected results with endpoint migration

At some point, you might decide to migrate endpoints from one gateway to another gateway for load balancing or other administrative needs. You can perform the migration if the destination gateway supports the protocol currently used by the endpoint.

If you have the **select_gateway_policy** defined, then the migration of the endpoint might not turn out as you would expect. For example, suppose you issue the **wep** command with the **migrate** option, specifying that the endpoint should migrate to gateway B. If you are migrating an endpoint from gateway A that is already down, then the **select_gateway_policy** script will supersede the **migrate** option of the **wep** command in this scenario.

During an isolation login, the endpoint manager receives the endpoint request to log in and uses the **select_gateway_policy** to determine which gateway the endpoint logs in to. Thus, the endpoint manager does not use what the **migrate** option of the **wep** command specified. In this case, you could have gateway C specified in the **select_gateway_policy**. With that, the endpoint manager assigns the endpoint to gateway C, instead of assigning it to gateway B, as expected.

To migrate endpoints successfully, add the desired gateway to the list in the **select_gateway_policy** script.

Running methods on the endpoint

You cannot use Tivoli Management Framework commands in method dependencies. There is no way to invoke these commands on the endpoint because there is no object dispatcher on the endpoint to run the commands. Instead of Tivoli commands, you should use frequently used commands such as shell, Perl, or bash that can be installed using file package rather than a dependency.

Moreover, only the **upgrade** option of the **wadminep** command is currently supported; all other remaining **wadminep** options are not supported. These options are better implemented by a task that can do several things at one time, for example removing a file followed by copying from another directory.

Mismatches between the endpoint manager and the name registry

The **wepmgr fsck** method on endpoint managers can rewrite Tivoli name registry endpoint records from the endpoint manager bdb files. This is a recovery convenience for those who have mismatches between the bdb files and the Tivoli name registry. The **wep** command with the **ls** option displays the endpoints listed in the bdb files. The **wlookup** command with the **–ar Endpoint** option displays endpoint records from the name registry.

To synchronize the Tivoli name registry from the endpoint manager bdb files, run this command:

```
wepmgr fsck
```

Failure to connect error message

The following is an example of the `Failure to Connect` error message:

```
date time +06: JOB THREAD EXCEPTION: ipc_create_remote failed:
unable to connect to 143.166.75.116+9494: (67) IPC shutdown
```

This error indicates that the gateway cannot find an endpoint at the IP address it expects (in this sample message, `143.166.75.116+9494`).

The following checklist can assist you in going through steps to identify the problem:

- Can the network of the endpoint reach the network of the gateway? Can both machines ping each other?

  Note: Both TCP and UDP communications are required.
- Was the endpoint daemon restarted on the endpoint? If not, then the gateway is never going to be able to talk to it. Communication is re-established at the request of the endpoint, not the gateway.
- Check that the port that the endpoint is using is the correct port. You can determine the port by examining the line in the lcfd.log file, which reads:
  ```
  successful bind to port +++
  ```
- What is the preferred gateway for the endpoint? Examine the last.cfg file.
- Was the connection to the gateway successful? Set the debugging level to 2 or higher with the **lcfd** command with the **–d***level*. At the bottom of the lcfd.log file, look for `Entering net_wait_for_connection`. This indicates that the endpoint is ready to receive commands from the gateway.
- This endpoint might have a name different than that used previously on the gateway. Use the **wep** command with the **ls** option to see all the endpoints. If you see both a *name* and a *name.*+++, it is likely that the only endpoint that is actually there is the one with the highest number at the end. These different names could have any combination of different IP address plus port numbers. They could all be different, could all be the same, or be a combination of different and matching numbers. For more information, see “Creating duplicate endpoints” on page 91.

″Timeout receiving method binary″ error message

When a gateway makes a downcall to an endpoint to spawn a method, the endpoint checks its cache for the files needed to run the method. If any files are missing, the endpoint requests the files from the gateway. The gateway then begins to send the files to the endpoint. By default, if the endpoint detects inactivity in this connection to the gateway for 20 seconds (for example, if the gateway is busy communicating with other endpoints), the endpoint closes the connection to the gateway. When this timeout occurs, the following error message appears in the endpoint log:

```
Timeout receiving method binary
```

This timeout value is a multiple of the **lcfd recvDataNumAttempts** and **recvDataTimeout** values. For example, the default value of **recvDataNumAttempts** is 10 and the default value of **recvDataTimeout** is 2, so 10 * 2 = 20 seconds. To increase the timeout value, increase the value of the **recvDataNumAttempts** or **recvDataTimeout** parameter.

Preventing a denial of service attack

A denial of service attack occurs when the endpoint receives a connection request that denies other processes the ability to communicate with the lcfd.

The endpoint has a single main waiting loop, in which it blocks on its socket (port) looking for a connection. When it receives a connection request, the lcfd places a block on that connection request indefinitely, and then waits for incoming data. If no data is forthcoming, the lcfd blocks input to the endpoint until either the process is stopped or the machine is rebooted again.

To prevent a denial of service attack, the endpoint maintains a queue of pending connections, which are connections that have been made but no data has been received on them. The endpoint cycles back to get the oldest pending connection and then reissues the timed received.

The **lcfd** command with the **–D** option reconfigures the endpoint during startup using the specified options. Configuration information is stored in the last.cfg file on the endpoint. There are three keywords that allow you to tune the behavior of the **lcfd** command with the **–D** option to prevent a DOS attack:

recvDataNumAttempts=*count*
: Specifies how many times to receive or attempt to receive data on the current connection before closing the connection. The default is `10`.

recvDataQMaxNum=*connections*
: Specifies how many connections to hold in the pending connection queue. A connection that is waiting for data is considered a pending connection and is added to the queue. After this limit is reached, all additional connection attempts are rejected until a pending connection is closed. The default is `50`.

recvDataTimeout=*seconds*
: Specifies how many seconds that a new connection waits for incoming data before requeuing the connection. If the pending connection queue is full, the connection request is rejected. The default is `2`.

Endpoints and policy regions

When you first install endpoints in your Tivoli region, these endpoints are not added to a policy region. The endpoints also are not listed on the Tivoli desktop in policy regions. To install an endpoint into a policy region, you can use the **winstlcf** command with the **–r** option. Another way to add endpoints to a policy region during installation is to use the **wln** command in the **after_install_policy** policy script.

Deleting endpoints

At some point during the life cycle of a Tivoli environment, there will be a need for you to delete endpoints. For example, you might install endpoints on machines in a prototype environment before you roll out Tivoli Management Framework to the enterprise. In this case, you might need to use the **wdelep** command to delete these endpoints. The **wdelep** command removes the specified endpoint from the Tivoli object database. You also can use the **wdelep** command with the **–d** option to delete the lcf.dat file from the endpoint system and shut down the lcfd process. The endpoint software, however, is not removed until you uninstall the endpoint. For platform-specific information on how to remove an endpoint from the Tivoli environment, see the section about deleting and uninstalling endpoints in *Tivoli Enterprise Installation Guide*

Chapter 5. Profiles, profile managers, and the profile database

In large distributed networks, machines are frequently grouped according to the type of work for which they are used. For example, machines in an engineering group might be used to produce Computer Aided Design (CAD) drawings, while those in an accounting group might be used to produce tax documents. With Tivoli Management Framework, you can place common configuration information for machines used for similar purposes in a centralized area. Doing so makes it easier to access, manage, and duplicate resources.

Profiles

Tivoli Management Framework provides a centralized management point to control data distribution to groups of systems. You can develop prototypes, known as *profiles*, distribute these profiles across a network, and apply them to a diverse set of machines. In the Tivoli environment, a profile provides a container for application-specific information about a particular type of resource.

The profile feature is also scalable. Information stored in a central location is distributed to numerous locations. You can affect many machines and several system configurations with a minimal amount of time and effort.

Profile Item Locking

You can lock individual items in a profile database to control which configuration elements can be modified by subscribers at lower levels. Locked profile items are read-only when distributed to lower levels in the hierarchy. In addition, when locked items are distributed, they override any local profile modifications. You cannot lock individual properties within a profile item.

For example, if the profile item for user Hunter is locked, subscribers at lower levels in the hierarchy can view the user properties for Hunter but cannot change them.

You can perform profile item locking either when a profile is initially created and populated or at a later time. Each profile item in a database contains, in addition to its value, the Locked option, which is used by the profile manager during profile distribution. If this option is set, the profile item cannot be modified by any subscriber. In addition, when locked items are distributed, they always override any locally defined modifications.

Profile Policy Support

Each profile contains a set of default policies and validation policies. *Default policies* define default values used when creating new profile items. *Validation policies* define permissible values for profile items. Establishing or modifying policies requires the **senior** role.

Profile policies can be constant values or scripts. Profile policies are stored in the profile database and share the following features with other profile configuration information:
- Policies are propagated from a profile to its subscribers.
- Policies can be resolved or run at the subscriber level.

- Local modifications to policies can be made at the subscriber level if you have the necessary authorization role.

Default and validation policies can be locked in the same manner as profile items. This enables you to define a profile policy so that the policy cannot be modified by a subscriber. As a result, a validation or default policy can be enforced throughout an installation.

If you have the required authorization role, you can modify a local profile policy and establish local rules for managing the systems you administer.

You can also disable policy validation on a profile. When validation is disabled, the validation policies are unchanged and remain stored in the profile database. However, the profile manager enforces these policies only if policy validation is enabled.

Default Records for Profiles

The default record for a profile specifies the default values used when a new profile item is added. The default values are profile-specific. Refer to the appropriate Tivoli application guide for information about profile default values for specific Tivoli applications.

To edit default policies for profiles, you must have the **super** or **senior** role.

You can define a default policy type for each profile item property. The defined policy can be one of the following types:

None	No default value
Constant	Default set as a constant
Script	Default set by a script

Validation Records for Profiles

Profiles can be validated to ensure that profile items comply with the current policy rules you have established. This ensures that lower levels in the hierarchy adhere to current policies.

The validation record for a profile specifies the validation policies used when a new profile item is added or an existing profile item is modified. To edit validation policies for profiles, you must have the **super** or **senior** role.

The defined validation policy can be one of the following types:

None	No validation value
Constant	Validation set as a constant
Script	Validation set by a script

You can also set validation policies on individual properties of a particular profile item. These validation policies can be defined as a constant or as a script.

For example, you can set a validation policy for the Accounting user profile. This validation policy might specify that all subscribers of this profile (for example, users in the Accounting department) have a home directory called /usr/home/*user_name* and that UNIX user IDs are between 4000 and 5000.

Profile Population

When you create a profile, it defines only the information that each profile item includes. The profile items must then be populated with the actual data that defines the particular system configuration. You can populate a profile with the appropriate information from the current system configurations at the targets. For example, to populate a user profile, you can populate the profile items with users from one or more targets.

You specify which targets to retrieve the information from and whether this information should overwrite existing profile information or be appended to the profile.

The ability to populate profiles enables you to model existing configurations in the network when a profile-based application is first installed. Typically, you might choose one or more targets that most closely represent the desired configuration and then use these targets to populate the profiles. When the profiles have been populated from the current configurations, you can make any necessary modifications to the profiles to bring them closer to the desired configuration. These profiles can then be subscribed to by targets and distributed to subscribers.

If an environment contains a target with thousands of resources (for example, a Network Information Services (NIS) user map with several thousand users), you should not create a single profile and populate it with thousands of items. Rather, you should determine a logical method for dividing the large number of resources into a subscription hierarchy. The advantage of creating such a hierarchy is that you can divide a large set of managed resources into smaller, more easily managed resource groups.

For example, if a system has several thousand users, you might divide them into groups based on department responsibilities. You can then create separate profiles for users in the Research department, the Grants department, the Administration department, and so on.

If you need to populate several profiles from a single large target database, validation policies can be helpful. If validation is enabled on a profile that is being populated, the populate operation filters out any target information that does not conform to the current validation policies. You are notified of any target information that does not conform to the current validation policies.

For example, if you want to include only users with user IDs between 300 and 800, you can specify this requirement in a validation policy and then enable validation. The profile population process incorporates only users with user IDs that meet the specified criteria. A list of any users that fail the validation process is displayed.

Modifications to a Profile

You can add items to a profile, modify the properties of existing profile items, and delete items from a profile. These profile operation are application specific. For information about adding profile items, modifying the properties of existing profile items, and deleting profile items from a profile, see the appropriate Tivoli application guide.

Synchronizing Profiles and System Files

If the system files and databases of a target are changed directly without using Tivoli Management Framework, the profiles that the target subscribes to might no longer accurately reflect the current configuration of the target

For example, suppose that the managed node Galileo subscribes to the user profile Pisa. You add new users Juliet, Leonardo, and Louisa to system configuration files on Galileo, rather than adding them to the user profile. As a result, the user profile Pisa no longer accurately reflects the configuration of Galileo.

You can synchronize profiles so that they accurately reflect the current configurations of the targets that subscribe to the profiles. When you perform a synchronization, you specify the type of profile to be synchronized. All profiles of the specified type are synchronized to their respective targets. You cannot synchronize an individual profile.

When you synchronize profiles, a list of differences between the profiles and the files and databases of their respective targets is displayed. Following are the types of differences:

- Profile items that exist in a profile database, but not in the files and databases of the target
- Items that exist in files and databases of the target, but not in the profile database
- Items that differ between files and databases of the target and the profile database itself

After the list of differences is displayed, you can confirm the synchronization. Items in a profile database that do not exist in files and databases of the target are deleted from the profile database. For items that differ between files and databases of the target and the profile database itself, the profile database is modified to match the files and databases of the target. For items that exist in files and databases of the target, but not in a profile database, you must choose which profile to add the items to. After the synchronization is completed, you can confirm any changes by viewing the profile items in the affected profiles.

Sorting Profile Items

You can change the display sort order for profile items by specified criteria. Profile items are initially sorted by a profile-specific default key. Profile items can also be sorted by an arbitrary set of profile-specific keys that represent profile fields. The profile items are sorted in the order of the keys and can be sorted in ascending or descending order for each key.

For example, suppose the default sort key for user profiles is login name, sorted in ascending order. Instead of using the default sort key, you might sort the profile items by primary group ID and then alphabetically by login name. In this case the sort key would be as follows:

1. Primary group ID, in ascending order
2. Login name, in ascending (alphabetical) order

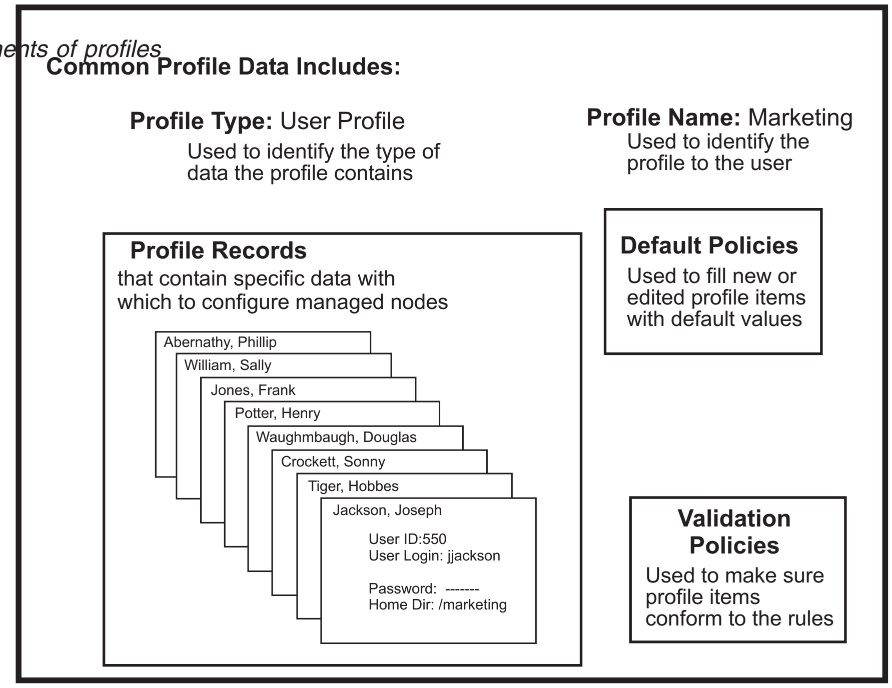
As a result, the profile items would be sorted first by primary group ID, and then each resulting group would be sorted alphabetically by login name.

Note: Specifying the sort order for a profile affects only the order in which items are displayed. The items in a profile will always be written in application-specific order. Refer to the appropriate Tivoli application guide for details.

Common Profile Elements and Functions

Although the content and functions are specific to a particular profile type, certain profile elements are common to all profile types. This section explains these common elements. For information about profile elements for a specific profile type, see the appropriate documentation for the Tivoli application.

Figure 1. Common elements of profiles


Common Profile Data Includes:
Profile Type: User Profile
Used to identify the type of data the profile contains
Profile Name: Marketing
Used to identify the profile to the user
Profile Records
that contain specific data with which to configure managed nodes
Abernathy, Phillip
William, Sally
Jones, Frank
Potter, Henry
Waughmbaugh, Douglas
Crockett, Sonny
Tiger, Hobbes
Jackson, Joseph
User ID:550
User Login: jjackson
Password: -------
Home Dir: /marketing
Default Policies
Used to fill new or edited profile items with default values
Validation Policies
Used to make sure profile items conform to the rules

All profiles contain the following information:

Profile type
Identifies the type of information the profile contains. For example, a user profile contains user default policies, user validation policies, and specific user information.

Profile name
Identifies the name of the profile. It is displayed as the profile icon label and is also displayed in profile-related windows.

Profile records
A set of records provide to a target with profile-specific information.

Default policies
A set of values applied to a new profile item.

Validation policies
Ensure that the profile records conform to the policies established in the policy region that contains the profile.

Profile View

You can view copies of profiles at all levels of the profile manager hierarchy. Lower-level profiles can contain records that have been locked at a higher level in the hierarchy. These profile records are read-only and cannot be edited.

The subscription hierarchy is displayed as a hierarchical view of the profile manager and all its subscribers. Each level of the hierarchy is indicated by an indentation. The greater the indentation of an item, the lower in the hierarchy the item is located.

```
- Profile Manager for General Admin
     - Profile Manager Office
           +Profile Manager Reception
           +NIS Domain Supply
     + Profile Manager Marketing
       Managed Node Finance
```

The first item displayed is the profile manager containing the original instance of the selected profile. In the preceding example, Profile Manager for General Admin is the containing profile manager. As you traverse the hierarchy, you can view profiles residing at different levels, as indicated by the indentation. Profile manager Profile Manager for General Admin has three subscribers: profile manager Profile Manager Office, profile manager Profile Manager Marketing, and profile manager Managed Node Finance. In addition, profile manager Profile Manager Office has two subscribers: profile manager Profile Manager Reception and profile manager NIS Domain Supply.

Profile Managers

Several profiles are required to describe the entire configuration of targets. Tivoli Management Framework provides profile managers to organize groups of profiles. Profile managers control the distribution of profiles to subscribers across an entire network or across a specified portion of a network.

Profile managers can operating in one of two modes: dataless and database. The mode determines the targets of profile distributions. You determine the mode in which a profile manager operates when you create the profile manager. You can change the mode by using the **wsetpm** command. For additional information about profile distribution based on operational mode, see the *Tivoli Management Framework Planning for Deployment Guide*. For information on the **wsetpm** command, see the *Tivoli Management Framework Reference Manual*.

Default Policies

Tivoli Management Framework uses both default and validation policies to control the operations of a profile manager. You can modify these policies to reflect your operational requirements. See the **wlspol**, **wgetpolm**, and **wputpolm** commands in the *Tivoli Management Framework Reference Manual* for procedures to modify a policy. The following default policies are provided:

pm_def_subscribers
Provides a list of potential subscribers to a profile manager. The list provided by this policy is displayed in the Subscribers window.

pm_def_profile_managers
Provides a list of profile managers to which a profile can subscribe.

Validation Policies

The following validation policies provided by Tivoli Management Framework govern profile manager subscription:

pm_val_subscribers
: Used by a profile manager to validate potential subscribers

pm_val_subscription
: Used by a profile manager or target to validate potential subscribers

pm_val_remove_subscribers
: Used by a profile manager to validate the removal of a subscriber

pm_val_remove_subscription
: Used by a profile manager or target to validate the removal of a subscription

To subscribe a managed resource to a profile manager, both the profile manager and the target must conform to the required subscription validation policies (**pm_val_subscribers** and **pm_val_subscription**). To remove a subscription, both the profile manager and the target must conform to the subscription removal validation policies (**pm_val_remove_subscribers** and **pm_val_remove_subscription**).

For example, if you want to subscribe managed node Ed to profile manager Wilbur, managed node Ed must conform to the **pm_val_subscribers** validation policies of profile manager Wilbur, and profile manager Wilbur must conform to the **pm_val_subscription** validation policies of managed node Ed.

For information about the profile manager policies, see **pm_val_subscribers**, **pm_val_subscription**, **pm_val_remove_subscribers**, and **pm_val_remove_subscription** in the *Tivoli Management Framework Reference Manual*.

Data Preservation During Distribution

During the distribution of select profiles, you can specify whether a distribution replaces or preserves local data:

Preserve Modifications
: Any local modification made to subscribers are retained

Make Exact Copy
: Subscriber profile records are completely replaced by the source profile records, creating a subscriber database identical to the source

Distribution Levels

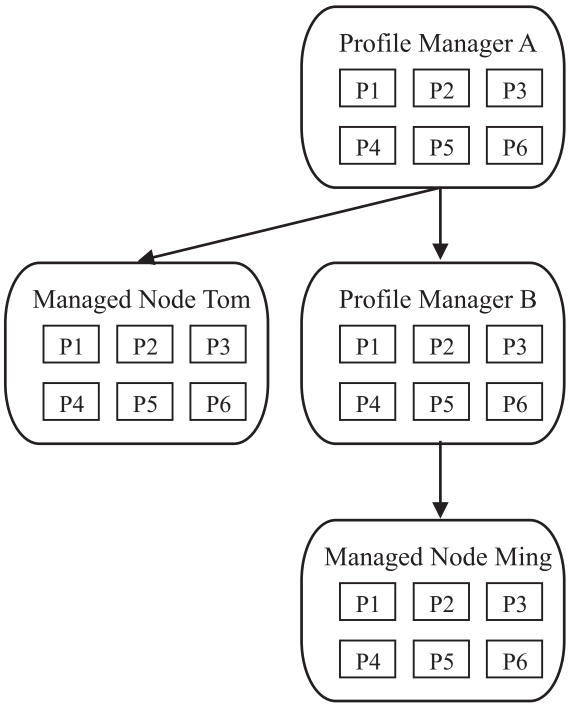
During the distribution of select profiles, you can specify one of two distribution levels:

Next Level of Subscribers
: Copies of the profiles are distributed only to immediate subscribers (that is, subscribers that are one level lower in the hierarchy)

All Levels of Subscribers
: Copies of the profiles are distributed recursively down the hierarchy to all targets

Profile managers and targets can reside at multiple levels in the distribution hierarchy. If you select **Next Level of Subscribers**, the distribution process affects only the next lower level in the hierarchy. You must perform the distribution process from profile managers residing at more than one level to reach all targets. Also, if you perform a distribution from a profile manager to a target using **Next Level of Subscribers**, only the target database is updated; the actual system files are not updated. You can then make any necessary local modifications in profile database on the target. You must perform a distribution from the target to affect the actual system files and databases.

In the following example, profile manager A has two subscribers, managed node Tom and profile manager B, and profile manager B has one subscriber, managed node Ming.



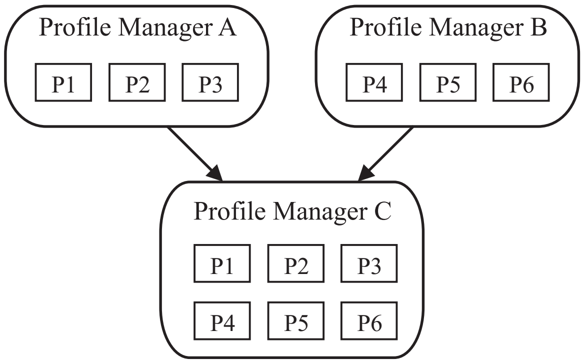
If you select **All Levels of Subscribers**, profile manager A distributes profile copies to managed node Tom, profile manager B, and managed node Ming.

If you select **Next Level of Subscribers**, profile manager A distributes profile copies only to profile manager B and managed node Tom. Managed node Ming, residing one level below profile manager B, does not receive the profile copies. For managed node Ming to receive the profile copies, the distribution must also be performed from profile manager B.

Recipient-based Profile Distribution

Getting a new copy of a profile is similar to distributing a profile except that the recipient, not the sender, requests the distribution.

When you get a new copy of a profile, a copy is distributed from the appropriate profile manager located one level higher in the hierarchy.



For example, profile manager C subscribes to profile manager A and profile manager B. Profile manager A distributes profiles P1, P2, and P3 to profile manager C. Profile manager B distributes profiles P4, P5, and P6 to profile manager C. If profile manager C performs a **get new copy** operation for profile P1, profile P1 is distributed from profile manager A.

Chapter 6. Tivoli Management Framework services

This chapter introduces some basic troubleshooting suggestions for Tivoli Management Framework services. Tivoli Management Framework consists of many services that are key to your ability to successfully manage systems in your environment.

This chapter explains how to perform tasks using the following services:

- "Tivoli Object Dispatcher (oserv)"
- "Tivoli Administrators and Roles" on page 117
- "Connected Tivoli Regions" on page 120
- "Task Library and Tasks" on page 125
- "Scheduler" on page 128
- "Notices" on page 128
- "Tivoli Desktop for Windows" on page 130

Note: For detailed descriptions of these services, see the *Tivoli Management Framework User's Guide*.

Tivoli Object Dispatcher (oserv)

The Tivoli object dispatcher, or oserv, is the central driving mechanism for operations in the Tivoli environment. Often, many of the problems you encounter with Tivoli Management Framework center around the functionality of the object dispatcher. Without the object dispatcher running correctly, Tivoli Management Framework cannot properly manage distributed computer systems. This section describes some common object dispatcher problems and their solutions.

It is important to remember that an object dispatcher process is running on the Tivoli management region server (Tivoli server) and on each managed node. There are some special considerations for Windows system accounts and user rights for those accounts. The object dispatcher starts as the Windows built-in system account because the object dispatcher is a service. When the object dispatcher starts, it attempts to validate the Tivoli remote access account. Moreover, several processes start as the local built-in administrator or **tmersrvd** account.

The oservlog file, described on "oservlog" on page 62, provides information about the object dispatcher. Simple tests to ensure that your object dispatcher is functioning properly include the following:

- If you are on the Tivoli server, ensure that you can run the following commands:
  ```
  odadmin odlist
  wlookup -ar ManagedNode
  ```
- If you are checking on the status of a managed node, enter one of the following commands to return information about the managed node:
  ```
  wping host_namewmannode node_name
  ```

  where *host_name* or *node_name* specifies the name of the managed node to be contacted.

Remember that an object dispatcher problem can occur on other systems besides the Tivoli server. If you suspect a problem with the object dispatcher, make sure that you determine which object dispatcher in the Tivoli environment is causing the problem. In addition, when diagnosing problems with the object dispatcher, try to answer these questions about the oservlog file:

1. What are the last few lines in the oservlog file?
2. Was the object dispatcher ever up?
3. What caused the object dispatcher to go down?
4. What changed in the environment since the object dispatcher last worked?
5. Verify that the network services are active and correct. You might need to check IP and host name lookups on your system.

Error Messages on Windows Operating Systems

You might encounter one or more of the following error messages on Windows operating systems:

Note: For a discussion about the Tivoli object dispatcher and its architecture, see the *Tivoli Management Framework Planning for Deployment Guide*

Error 1067

If you enter `net helpmsg 1067` from a command prompt, the following message is displayed:

```
The process terminated unexpectedly.
```

This message indicates that the object dispatcher did not start as a service. To correct this error, examine the last few lines in the oservlog file to determine the cause of the problem.

date time: CreateWindowStation failed Access is denied.

This message indicates that the permissions for the **\system32** directory are incorrect. To correct this error, ensure that the `Bypass traverse checking` user permission contains the **tmersrvd** account. To do this, click **Start → Programs → Administrative Tools (Common) → User Manager** to view the User Manager window. Next, click **User Rights** from the **Policies** menu to ensure that `Bypass traverse checking` contains **Everyone**. Another way to correct this error is to run the $BINDIR/TAS/INSTALL/ ntconfig.exe file from the database directory on the local system. This adds the new information to the **tmersrvd** User Rights policy.

date time: CreateWindowStation failed: The specified module could not be found.

This message indicates that there is a problem with system resources. To correct this error, click **Start → Settings → Control Panel → Keyboard → Input Locales** and ensure that the English (United States) keyboard layout is added to the list. Note that the English keyboard does not have to be the active keyboard.

date time: !could not find oserv-h.exe

This message indicates that the permissions for the **\system32** directory are incorrect. To correct this error, ensure that the `Bypass traverse checking` user permission contains the **tmersrvd** account. To do this, click **Start → Programs → Administrative Tools (Common) → User Manager** to view the User Manager window. Next, click **User Rights** from the **Policies** menu to ensure that `Bypass traverse checking` contains **Everyone**.

Another solution is to run the $BINDIR/TAS/INSTALL/ ntconfig.exe file from the database directory on the local system. This adds the new information to the **tmersrvd** User Rights policy.

This message also occurs in the following situations:

- When the oserv.exe file is not in the database directory
- When a restore operation is performed on a Windows operating system and the oserv.exe file was not moved to the directory

date time: tap_get_sid_logon_token failed: The system cannot find the file specified.

This message indicates that the TivoliAP.dll file does not exist or is corrupted. To correct this error, follow these steps:

1. Reinstall the TivoliAP.dll file if it has been deleted from the **\system32** directory. (You also can copy it from the **BINDIR\tools** directory.)
2. To properly set up this file, enter the following command:

   ```
   wsettap -ar account
   ```

   Note: This step is optional unless you want to access shares, which are not accessible on this machine.

You do not have to reboot your Windows operating system unless you delete or add the TivoliAP.dll file to the local security authority (LSA). If you run the **wsettap –d** command to delete or run **wsettap –ar** " " to add it for the first time, you also must reboot the system.

date time: tap_get_sid_logon_token failed: Logon failure: the user has not been granted the requested logon type at this computer.

This message indicates that the **tmersrvd** account or the **Everyone** account does not have the `Log on locally` user permission. To correct this error, click **Start** → **Programs** → **Administrative Tools (Common)** → **User Manager** to view the User Manager window. Next, to ensure that `Log on locally` contains **tmersrvd** or **Everyone**, click **User Rights** from the **Policies** menu.

date time !tap_call_init_ failed

This message indicates that you are not using a valid user name and password for the computer. To correct this error, use the **wsettap** command to properly map the valid user to the Tivoli remote access account. The following variations of this error message might appear:

date time: !tap_call_init failed: A specified authentication package is unknown

This message indicates that either Tivoli Authentication Package was not added to the LSA chain in the registry or that the system was not rebooted after it was installed as a managed node. To correct this error, enter one of the following commands to set the access in the registry and then reboot the system:

- `wsettap -ar account`
- `wsettap -ar ""`

date time: !tap_call_init failed Error 2221

If you enter `net helpmsg 2221` from a command prompt, the following message is displayed:

```
The user name could not be found.
```

To correct this error, use the **wsettap** command to check the TRAA and ask yourself, "Is this a valid user name and password on this system?" Issuing the **net user** command lists the local accounts. Use the **wsettap –r** *account* command to set it to a valid account or, if you do not need the account, enter the following command to delete it:

```
wsettap -r " "
```

date time: !tap_call_init failed Error 2453

If you enter `net helpmsg 2453` from a command prompt, the following message is displayed:

```
Could not find domain controller for this domain
```

This message indicates that the domain controller could not be found and the system is unable to access the primary domain controller (PDC). The API Tivoli Management Framework uses only accesses the PDC. To correct this error, use the **wsettap** command to check the Tivoli remote access account. Next, issue the **net user** command to list the local accounts. You can use the **wsettap –r** *account* command to set it to a valid account or, if you do not need the account, enter the following command to delete it:

```
wsettap -r " "
```

date time: fork failed (errno 22) date time: @CreateProcessAsUser failed: Access is denied

This message indicates that the permissions on the oserv-h file are incorrect. To correct this error, ensure that the permissions on this file are **Read** and **Execute** for the **tmersrvd** account.

date time: !oserv: odlist init failed. requested resource not found. (30)

This message indicates that there is a problem with system resources. Causes of this message include the following:

- The odb.adj or odlist.dat file is missing from the database directory or is corrupted. To correct this error, restore the managed node or Tivoli server from a backup.
- The object dispatcher failed to get a port above 1023. This happens when the port range is set improperly with the **odadmin set_port_range** command. If possible, examine the output of the **odadmin odlist** command. To see if the port range on Windows is restricted, click **Settings** → **Control Panel** → **Network** to display the Network notebook. Next, click **Protocols** (TCP/IP Protocol) → **Properties** → **IP address** → **Advanced** → **Enable Security** check box → **Configure**. The TCP/IP Security window is displayed.

Another solution is to change the permissions on the **DBDIR** and **DBDIR/file_versions** directory to give everyone total access. It is also a good idea to check for the required files in the database directory, such as odb.bdb, odlist.dat, imdb.bdb, notice.bdb, and file_versions.

General Error Messages

You might encounter the following error messages on any supported platform:

date time: get_boot_info failed (30)

This message indicates that the odb.adj file is missing from the database directory or is corrupted. To correct this error, restore the managed node or Tivoli server from a backup. For more information, see the **set_ORB_pw** option of the **odadmin** command in the *Tivoli Management Framework Reference Manual*.

date time: odlist init failed. named resource already exists

This message is present in the oservlog file, where you attempted to install the managed node. It indicates that you attempted to register a managed node on a Tivoli server where it was already registered. To correct this error, you must remove all references to the managed node from the Tivoli management region (Tivoli region) and reinstall the managed node. For more information, see the **wrmnode** command in the *Tivoli Management Framework Reference Manual*

To correct this error on a NetWare gateway, follow these steps:

1. From the command line of the Tivoli server, enter the following command to remove the managed node:

   ```
   wrmnode host_name
   ```

   where *host_name* specifies the host name of the NetWare system.
2. If the managed node was not registered, enter the following command:

   ```
   odadmin odlist rm_od dispatcher_id
   ```

   where *dispatcher_id* is the ID of the dispatcher number running on the NetWare gateway.
3. To clean up the object database on the Tivoli server, enter the following command:

   ```
   wchkdb -u -x
   ```
4. If the binaries are already installed, you do not need to reinstall them, but you do need to delete the contents of the **$DBDIR** directory on the NetWare system before reinstalling the NetWare gateway.
5. Run the following command:

   ```
   SYS:TIVOLI\BIN\NWR-IX86\BIN\OSERV1ST -s installation password
   ```

   where *installation_password* is the installation password if one was defined.
6. Run the following script from the Tivoli server to register the managed node with the Tivoli name registry:

   ```
   bash nw_TMF_Install.sh host_name
   ```

   where *host_name* is the TCP/IP host name of the NetWare system.

date time: !oserv: odlist init failed. could not encrypt/decrypt data. (43)

This message indicates that there is a problem with the information necessary to encrypt or decrypt data in the oserv-to-oserv communications. To correct this error, save the odb.adj file on the managed node and restore the file from a backup of that managed node. You also can recreate the odb.adj file by following these steps:

1. To ensure that the managed node was not registered, enter the following commands:

   ```
   wlookup -ar ManagedNode
   odadmin odlist
   ```
2. To create a *managed_node-od* #–odb.adj file in the database directory of the Tivoli server, enter the following command:

   ```
   odadmin set_ORB_pw od
   ```

   where *od* is the object dispatcher number.

   Note: This works only if the Tivoli server and managed node are both UNIX operating systems or both Windows operating systems.
3. Copy the newly-created binary file to **DBDIR/odb.adj** on the managed node and restart the managed node.

To verify the installation password, enter the following command:

```
odadmin set_install_pw
```

For the old key value and the new key value, enter what you believe to be the old key. If you receive a `rls:invalid argument specified` error message, stop the object dispatcher and start it using the **oserv –s** *install_key* command.

Note: To correct this error on NetWare systems, see the section about problems with installing the NetWare gateway in the *Tivoli Enterprise Installation Guide*

date time: !oserv: odlist init failed. host inaccessible or host failure. (48)
This message indicates that there is a host name resolution problem from the managed node to the Tivoli server. To correct this error, ensure that you can resolve your Tivoli server name from that host.

date time: $Database mismatch date time: !oserv: odlist init failed. internal resource corrupted. (54)
This message indicates that the object dispatcher is attempting to initialize the list of dispatches and a required resource is corrupted.

Note: Ensure that you source the appropriate setup file and that your system environment is set correctly.

To correct this error, follow these steps:

1. On Windows operating systems, set $**NDIR** to $**BINDIR**. (Do not add blank spaces after the forward slash (/) symbol).
2. Enter the following command:
   ```
   net start oserv /-b%NDIR% /-Nali
   ```
3. Enter the following command:
   ```
   net start oserv /-b%NDIR% /-Nali /-B%LIBDIR%
   ```
4. Manually perform a restore of the database.

date time: @od_init: Unable to Establish Connection to ALI
This message indicates that the managed node cannot contact the Tivoli server. In this instance, the managed node cannot resolve the name it has for the server. To correct this error, ensure that there are no communication problems between the managed node and server. If the host name or IP address of the server has changed, see "Changing IP names and addresses" on page 16.

date time: ipc_accept failed: IPC shutdown
This message indicates that the object dispatcher is shut down while the Tivoli desktop is running and that an interprocess communication (IPC) function has failed. To correct this error, restart the object dispatcher service. The **wtrace** command output displays the time of failure.

date time: !oserv#Undo conflict. Bad locking protocol
This message indicates that a transaction is locked by a method. To resolve this error, run the **tmstat** command to display which transaction is locked. This error can also indicate that two indicators are open in one indicator collection. This problem can be solved after a ruleset modification to the event or monitoring programs.

date time: Database corruption error 34
This error indicates that there is a problem with system resources. To correct this error, check to see if the /etc/hosts file has control characters in the file and delete them. This is usually a problem where the label of the Tivoli region differs to what the resolvers hand back.

date time: Cannot Map Address

This message indicates that the object dispatcher received a request from an IP address, which was not registered in **odadmin odlist**. This message is common with multi-interface systems or systems that have recently changed their IP address. It also can occur when systems are trying to break in.

This error message is logged on the system, which received a request. For security reasons, the calling system is not notified that there was an error in the request. This error message is most likely to appear on the Tivoli server, but can also appear on managed nodes, especially managed nodes that also are application services, such as the RIM host and event server.

If it is determined that the host name that maps to this IP address is a valid managed node, follow these steps:

1. To determine which system contains the object dispatcher referred to in the error message, enter the following command:
   ```
   odadmin odlist
   ```
2. Determine the correct IP address for the managed node or Tivoli server.
3. To correct the IP address error in the Tivoli object database, enter one of the following commands:
   - `odadmin odlist change_ip`
   - `odadmin odlist add_ip_alias`

   For valid **odadmin** parameters, refer to the **odadmin** command in the *Tivoli Management Framework Reference Manual*. Examples of the **odadmin odlist** command also are provided in “Changing IP names and addresses” on page 16.

Tivoli Administrators and Roles

The Tivoli environment uses administrators to delegate the use of the system accounts without giving those administrators the system password or complete control.

In addition, Tivoli Management Framework requires certain accounts to be set up and to remain on UNIX operating systems and Windows operating systems. During installation, these accounts are created on the systems where Tivoli Management Framework is installed. This account is used for most Tivoli Management Framework operations. You might encounter problems if these accounts have been changed or deleted. Use the following list for these accounts. The account varies with each operating system as follows:

HP-UX
tmersrvd:*:59999:59999:Reserved for TME:/:/bin/false

AIX nobody:*:4294967294:4294967294::/:

Solaris
nobody:*:60001:60001::/:

On Windows operating systems, Tivoli Management Framework requires the **tmersrvd** user account and the **Tivoli_Admin_Privileges** group account. These accounts are created automatically when the Tivoli server, managed node, or endpoint is installed. If the user account **tmersrvd** is deleted, you must run the $BINDIR/TAS/INSTALL/ntconfig.exe file from the database directory on the local system to recreate the required Tivoli account. The ntconfig.exe file has the following functionality:

- It runs as the user defined in the Default Access Account.
- It is responsible for creating the **tmersrvd** account and the **Tivoli_Admin_Privileges** group account.
- It assigns the required user rights.

Tivoli Management Framework requires special accounts to manage NetWare gateways and endpoints. The following accounts are needed when you use Novell Directory Service (NDS):
- For endpoints, **Admin** (for **root**) and **lcfrsrvd** (for **nobody**)
- For gateways, **Admin** (for **root**) and **tmersrvd** (for **nobody**)

If you use bindery emulation instead of NDS, the supervisor account is needed instead of the **Admin** account.

The following section provides a few suggestions for working with administrator access.

Removing versus Deleting Administrators

When you remove an administrator from the Administrator window, the administrator is removed from the Tivoli desktop, but not the database. To permanently remove an administrator, you must delete it.

To find out whether or not you removed an administrator instead of deleting it, use the **wlookup –ar Administrator** command to list administrators defined to Tivoli Management Framework. Next, list the administrators in the administrator collection. If there is a difference, use the **wln** command to link the required administrator object back to the **Administrators** collection, as in the following example:

```
# wlookup -ar Administrator
Root_blue 1999200098.1.192#TMF_Administrator::Configuration_GUI#
Viki      1999200098.1.566#TMF_Administrator::Configuration_GUI#
root@blue 1999200098.1.192#TMF_Administrator::Configuration_GUI#
viki      1999200098.1.566#TMF_Administrator::Configuration_GUI#

# wls /Administrators
Root_blue     ! Viki is defined but not in the collection

# wln @Administrator:Viki /Administrators
```

After the administrator is linked back into the GUI, you can delete it using the **delete** option from the Tivoli desktop or the **wdel** command.

Root Authority

By default, the Tivoli root administrator has root authority. This account possesses full unrestricted privileges for accessing and managing systems. For example, one such privileged action is being able to assign roles to administrators even if the root administrator does not have the role himself. On UNIX operating systems, the Tivoli root administrator account has UNIX root permissions; on Windows operating systems, the account has Administrator permissions.

A Tivoli server can have multiple root administrators. You can encounter problems with the ability to grant roles if you decide to revoke root authority to all administrators in the Tivoli region. As a precaution, you should always have at least one administrator set up having root authority. Before deleting global roles for the Tivoli root administrator, you should use the **wauthadmin** command to ensure that another administrator has root authority at all times. You should have at least one administrator with root authority (set by the **wauthadmin** command). If you choose to delete the Tivoli root administrator, another administrator with root authority can grant roles as required. You cannot grant roles that you do not have unless you are the root administrator on the Tivoli region where you are granting the roles.

Determining if an Administrator Has the Correct Permissions

The layers of checking used to determine whether an administrator has permission to perform an activity include the following:

1. An administrator is authenticated as a Tivoli administrator. This only happens within the local Tivoli region. Each administrator who performs tasks must be defined in the Tivoli region where they perform the task.
2. An administrator is authorized to perform the activity based on the authorization roles.
3. An activity is performed as the user and role specified in the method. This requires the correct user, group, and roles to be defined.

Whenever there is evidence of permission problems, use these steps to determine at what level the permissions are not correct.

Hints for Troubleshooting Administrators

The following list summarizes additional items to consider when working with administrators:

- You should avoid giving remote root access. Always specify a host name if you want to allow a root user to use a Tivoli administrator name (use a name such as **root@hostx.domain**).
- If you set a login name, Tivoli Management Framework does not check the validity of the user name when the administrator is created or modified. There is no check that the user actually exists until a method needing the user name is executed.
- You will have to subscribe the relevant administrators to notice groups as new applications are installed.
- Roles are not hierarchical. They must be selected and used individually.
- Roles do map directly across connected Tivoli regions, except **super** for the Tivoli region, which maps to **user** for the remote Tivoli region.
- For a Windows administrator user ID, you should also enter the Windows domain when setting a login name. For example:
  ```
  domain/user@managed_node
  ```
- You should not try to set the same current login name (through the Set Login Name window) for more than one administrator.
- The administrator must have the proper role to perform the activity.
- If trying to perform something out of Tivoli Management Framework, such as write a file, ensure that the Tivoli administrator has the appropriate operating system account permissions.
- `BAD_ID` in the error message means the account name is not valid on the target on which the method is being executed.
- `UNAUTHORIZED` could mean that the logon is not valid or the authorization role is not high enough to perform the task.

Connected Tivoli Regions

In some instances, the Tivoli environment contains several Tivoli regions that are connected to each other. The administration of connected regions can be complex if you do not have a good diagram of the connections between the regions. Before you begin troubleshooting, you should create a diagram of the connections between Tivoli regions in the Tivoli environment. This diagram can assist you in determining the locations of your Tivoli regions and the types of connections in your Tivoli environment.

The following section discusses problems encountered with interregion passwords, secure and remote connections, one-way and two-way connections, connecting and disconnecting, resource exchange, resource visibility, actions on remote Tivoli regions, and application failures.

Interregion Passwords

When installing the Tivoli Management Framework, you are prompted for the installation password. This password also becomes the interregion password, as well as the intraregion password. If no password is entered for the installation password window, a password is still used by Tivoli Management Framework. However, it is a hidden password that cannot be recovered if the user changes or tries to make the interregion password null. This hidden password can cause problems with later attempts at performing interregion connections on this Tivoli region. Finally, if you click inside the installation password window during installation but do not enter a password in this field, the passwords are effectively nulled. This causes interregion connection issues that can seem to have no cause. In any case, the best solution is to set the passwords on all Tivoli regions that will be interconnected using the **oserv –R** command. You should also run the **oserv** command to ensure that all encryption levels are set properly on all Tivoli regions that will be interconnected by running the **oserv –C** command.

Ensure that the interregion encryption levels are set correctly on both sides by using the **odadmin region set_region_crypt_level** command. Use the **odadmin** command to determine the encryption levels on Tivoli regions before proceeding with interconnects.

The interregion password is initially set from the installation password. After the Tivoli region is in place, setting the **odadmin set_region_pw** command affects the interregion password.

Before you change the interregion password using the **odadmin set_region_pw** command, you must first disconnect the connected Tivoli regions.

If you change the interregion password while the Tivoli regions are still connected, interconnected activity fails. If this happens, disconnect the regions using the **wdisconn –s** command, and then reconnect the regions using the new password. If you cannot disconnect the regions, restore the passwords in the connected Tivoli regions to the previous password.

SSL Network Security and Connected Tivoli Servers

The creation of an inter-region connection is not supported when the network security setting on a remote Tivoli server is set to FORCE_SSL. Therefore, you must temporarily set the network security setting on the remote Tivoli server to SSL using the **odadmin set_network_security** command until the inter-region connection is successful. After you complete the inter-region connection, you can reset the the network security setting on the remote Tivoli server to FORCE_SSL.

Secure and Remote Connections

A remote connection allows one Tivoli region to provide all of the connection request information and to make the connection. Remote connections perform the same steps as a secure connection, but the **rexec** command or **rcmd** command is used to communicate to and start the connection in the remote Tivoli region. To perform remote connections, the Tivoli region making the connection must have the root password of the remote Tivoli server or be a trusted host.

Note: The /etc/hosts.equiv and .rhosts files need to be properly configured if a connection is attempted with a trusted host facility. Remember that /etc/hosts.equiv does not affect the root user; if the interconnection is made by root, /etc/hosts.equiv has no effect.

For Windows operating systems, you cannot use the trusted host facility; you must use Tivoli Remote Execution Service with the shell service option. Because it is possible to disable the Tivoli Remote Execution Service after the initial installation, you might need to check that this service is installed and running to perform the Tivoli region connection.

A secure or single-sided connection does not require a remote login to connected Tivoli regions. If creating a secure interconnection, the process must be completed on both hosts to create a two-way interregion connection. This method protects the possibility of a root or administrator password from traveling across the wire.

One-way and Two-way Connections

One- and two-way connections determine the visibility of the resources of interconnected Tivoli regions. A one-way connection requires one of the Tivoli regions to act as the manager of the resources for the other Tivoli region. The managed Tivoli region will not be able to view the resources of the managing Tivoli region.

This type of connection can cause applications to experience problems. For example, distributed monitoring can be set up so that monitors will change indicator collections on the other side of a connection. In a one-way connection scenario, the remote managed Tivoli region where the monitor is running cannot see the indicator collection resources in the local managing Tivoli region.

A two-way connection allows each interconnected Tivoli region to view all resources as long as they are updated regularly. When you encounter problems connecting Tivoli regions, do not update resources. You should perform all updates after the regions are connected so that there is one less item to troubleshoot with the connection process.

Unable to Connect to a Previously Connected Tivoli Region

If there was a two-way connection between Tivoli region A and Tivoli region B and you are now unable to connect the previously connected Tivoli region, the following suggestions can assist you in re-establishing the connection:

- Issue the **wlsconn** command on Tivoli region A to see if there is still a connection to Tivoli region B. Log in to the Tivoli server in region B and do the same. If only region B shows that the connection is active, issue the **wdisconn** command with the **–s** option from region B.

- Use the **wchkdb –ux –o** *outfile* command after the disconnect has completed. The **–o** option stores references to problem objects in a file. Subsequent checks can then use that file as input rather than rechecking every object.

If errors occurred during the check, you can try taking some corrective actions before running the **wchkdb** command again. This time, however, replace the **–o** option with the **–f** option. This speeds up the check by just looking at the objects listed in *outfile*. The next **wchkdb** command will show if the errors persist. There might be occasions when running the check more than once is enough to resolve the errors. This is the format of the **wchkdb** command to use when specifying an input file name:

```
wchkdb -ux -f outfile
```

For more information about the **wchkdb** command, see "Checking Database Consistency" on page 52.

Unable to Disconnect a Tivoli Region

If only one side shows a connection, you can try a one-sided disconnect from the Tivoli region where the connection appears active, as shown below:

```
wdisconn -s region-name
```

If disconnecting Tivoli regions fails, check if both regions are showing a connection by using the Tivoli desktop or the **wlsconn** command and issuing the **odadmin region** command. If Tivoli region B does not have the Tivoli region A region listed in the output from the **odadmin region** command, enter the following command on region B to manually add the region back:

```
odadmin region add_region region host port [crypt]
```

You are also prompted for a password, which should be the same encryption password that you used for the initial connection. When you have added the region manually, you can reissue the **wdisconn** command from Tivoli region B.

Updating Name Registry Performance

The following sections describe ways to update name registry performance.

Verifying the Name Registry

Resource exchanges between connected Tivoli regions should be performed on a regular basis to ensure that all name registry information is current. How frequently you exchange resources depends on how frequently you create, modify, or delete Tivoli resources in a given Tivoli region. You should determine a schedule for resource exchanges based on how often Tivoli resources are created, modified, or deleted in your Tivoli environment.

Use the **wchkdb** command to verify that the name registry does not contain any references to deleted objects and to verify that all objects that should have name registry references do have references. See "Repairing the Tivoli Database" on page 52 for information about verifying the name registry.

Changing the Size of the Name Registry Cache

You can change the size of the name registry cache to optimize Tivoli interregion performance. The default value for the name registry cache is 16 pages per resource, and a page is 8 KB.

Follow these steps to determine the current size of the name registry cache and to change the size of the cache:

1. Log in to a managed node or the Tivoli server on which your Tivoli administrator has an alias with the **super** role for the Tivoli region.
2. Source the appropriate setup file to ensure that your system environment is set correctly. See “Setting Tivoli environment variables” on page 1 for details.
3. Enter the following commands with the options specified to get the current size of the name registry cache:
   ```
   TNR=`wlookup NameRegistry`
   idlattr -t -g $TNR cache_pages long
   ```
4. To set the size of the name registry cache to a new size, enter the following commands with the options as specified:
   ```
   TNR=`wlookup NameRegistry`
   idlattr -t -s $TNR cache_pages long num
   ```
   This changes the size of the name registry cache to the number of pages specified by *num*.

The size of the name registry cache represents a compromise between speed of lookups and system resource usage. Generally, a larger cache results in better Tivoli interregion performance; however, a larger cache reduces the amount of system memory available for applications. The default setting is adequate for typical environments with a reasonable number of resources managed by a Tivoli server configuration, as recommended in the *Tivoli Management Framework Planning for Deployment Guide*. See the *Tivoli Management Framework Reference Manual* for more information about the **idlattr** command.

Updating Resources

Updating resources can be a fairly demanding process. Running as a single transaction, the update method obtains a write lock in the local name registry for any resource types that are updated and sets a read lock for resources in remote Tivoli regions. If the update method runs for any significant amount of time, other methods are locked out of portions of the name registry until it completes. In remote Tivoli regions, where only read locks are held, this might not be a problem. However, in the local Tivoli region where write locks are held, even **wlookup** commands are blocked. This blocking can cause severe performance problems.

The following are rules for updating resources:

- Do not schedule updates too often. When a Tivoli environment reaches a steady state, there is no reason to update resources more than twice a day. Once a day during non-peak hours is preferable. An immediate update of a single resource type can be forced with the **wupdate** command.
- Do not schedule updates in interconnected Tivoli regions at the same time. Stagger scheduled updates at least 30 minutes apart.

See the *Tivoli Management Framework User’s Guide*for more information about Tivoli regions and the name registry.

When Remote Resources Are Not Visible or Accessible

You might encounter problems with resources not being visible with connected Tivoli regions. One instance of this could be when Tivoli region A and Tivoli region B are connected and a managed node is added to region A. Until a resource update occurs between region A and region B, region B will not have the information to effectively manage or view the new managed node. The same problems can occur if a resource is changed or deleted. Thus, a name change to a resource in region A would not be visible in the region B name registry until a resource update is completed.

Resource visibility (through the name registry) issues can occur in the following situations:

- Available lists, as in Available Subscribers in a Profile Manager window
- Resources listed with the **wlookup** command
- Resources referenced from the command line with **@Resource:instance**

One caveat to visibility pertains to collections that contain resources in remote Tivoli regions. There are instances where you can still see resources although they are not listed in the Tivoli name registry of the remote Tivoli region. These instances include the following:

- Resources viewed by opening collections on the Tivoli desktop. Collections here refer to any kind of a container object that is visible from the Graphical User Interface (GUI).
- Resources managed through the file-system-type commands (**wls**, **wmv**, and so on).
- Resources referenced from the command line with **/xxx/yyy**.

If only one Tivoli region is seeing shared resources, consider the following items:

- Remember that the exchange of resources is a *pull* operation. The Tivoli region not seeing any remote resources has probably not issued a **wupdate** command to the other Tivoli region.
- If you have a one-way connection, only the managing Tivoli region will see the remote resources. The manager option can be selected during a secure connection or the managing Tivoli region will be the one from which the remote connect was started.

Another resource visibility problem can occur when there are firewalls between the connected Tivoli regions or between the resources and the Tivoli regions. Ensure that the firewall configuration does not filter the port range you are using for communication between the Tivoli regions.

Unable to Perform Actions on Remote Objects

After you have established that the resources are visible in the connected Tivoli region, your administrator can have an access problem to remote resources. The problem might be that your administrator user ID does not have the authority to operate on objects in policy regions in the remote Tivoli region. This can be achieved by the following ways:

- Give the administrator the correct level of access across the policy regions in the remote Tivoli region. Update the **TMR Roles** option when updating or creating an administrator.
- Select specific roles for selected resources using the **Resource Roles** option when creating or updating an administrator.

You cannot update the roles for resources that you cannot manage yourself. You cannot update the resource roles of an administrator that has higher privileges than you.

Application Failures across Region Boundaries

You might encounter problems with Tivoli applications failing when running them across Tivoli regionboundaries. The following describes two items you can review for application failures:

- Check that you have a two-way connection. Many applications that run across a connection need to return information to the Tivoli region from which they were started. If the process and information can only flow one way, the application might not function correctly.
- Check that the application is installed on Tivoli region A, Tivoli region B, and all the managed. For example, running an inventory scan on Tivoli region A for resources in Tivoli region B requires that the inventory product be installed on each Tivoli server and all of the machines you want to scan in both Tivoli regions.

Task Library and Tasks

Before you can create a task library and its components, the policy region must have **TaskLibrary** as a managed resource type. When you create a task, the required role defined in the task is the role that you have assigned in the policy region where the task endpoint resides, not in the policy region where the task library is located. For example, suppose you have the **admin** role in the policy region where the task library is, but only the **user** role in the policy region in which the task is executed. Assuming that you have no Tivoli region roles, the role required to run the task must be **user** to be able to run the task in that policy region. If you have a Tivoli region role specified, that role could be used as the required role for the task.

If you have updated the executable file for a task and find that the old executable file is still running instead of the new one, you must re-specify the executable file in the task definition so that it can be refreshed in the Tivoli server binary tree. You can do this either with the **wsettask** command or from the Tivoli desktop.

To create a log file for tasks, set the TivRepoLog environment variable using the **odadmin environ set** command. Set the environment variable as follows, where *value* is a relative or fully qualified path and file name:

```
TivRepoLog=value
```

This log file contains the following information:
- The name of the task
- The task library in which the task is stored
- The name of the user or group under which the task is run
- The approximate size of the data that is returned by the task

Note: The user or group under which the task runs must have read and write permissions for the directory and file specified for TivRepoLog. Otherwise, the task will not run.

The following tips can be useful in investigating problems with tasks:
- Be sure that your script or binary file is working as it should be, independent of the task or job process. You can do this on UNIX systems by opening an xterm window on the machine where the problem task is running by entering the following command:
  ```
  wxterm -h ManagedNode -display mydesktop:0
  ```
  This gives you the xterm window and confirms that remote initiation of programs is possible. The xterm window will run with the same environment as the task.
- Normally, the first line of a task script must be **#!/bin/sh**. However, if the task is to execute on an endpoint, you can omit this line.

- The binary directory of the tasks must be writable for creating new tasks. The following is the path for this directory:

  ```
  /$BINDIR/TAS/TASK_LIBRARY/regionnumber/bin
  ```
- To change the validation policy for a policy region so that it allows root to run tasks, follow these steps:
  1. Enter the following command:

     ```
     wcrtpol -v TaskLibrary new_name_for_library_copy
     ```
  2. Enter the following command:

     ```
     wgetpolm -v TaskLibrary new_name_for_library_copy \
     tl_val_set_uid >file_name
     ```
  3. Edit the *file_name* and remove the checks for root. Keep the part that says: `echo TRUE and exit 0`.
  4. Enter the following command:

     ```
     wputpolm -v TaskLibrary new_name_for_library_copy \
     tl_val_set_uid <file_name
     ```
  5. In the Policy Region window, open the Managed Resource Policies window. Select the new validation policy from the drop-down list.
- If you are using commands from Tivoli Management Framework in your task, you must set up the environment variables in the script with setup_env.sh or setup_env.cmd, as appropriate.
- Tasks that are run with the user name left blank need to run under the special Tivoli account. If there are problems with these tasks, ensure that the account still exists on the target system. See "Tivoli Administrators and Roles" on page 117 for more information about the special Tivoli account.
- Tasks that are run with the user name or group name set to `'*'` will run as the administrator who is executing the task. If an administrator does not have an account on the target system or does not have the authority to execute or fulfill the task, the task will fail to execute, give an error, or appear to not work. Either provide the administrator with the appropriate access to the target system or run the task with no user listed in the task properties (the task runs as the Tivoli account).
- Find out if the task associated with the job still exists by using the following command. The tasks and jobs within the library are displayed.

  ```
  wlstlib library_name
  ```

This list describes some of the more common errors that occur with the task library:

time out
: The task exceeded the amount of time allowed in the timeout setting for the task. If the task is for an endpoint, the task might have exceeded the gateway session timeout as well. The default is 300 seconds. To configure the gateway session time outs, use the **wgateway set_session_timeout** command. Note that exceeding the time out value does not end the task on the target. It only fails from the point of view of the server.

getpwname failed with code ##
: You are trying to execute the task with a user that does not have an account on the destination managed node where it is trying to run the task. The user must exist before the task can be run.

method fork failed
: This error can be an operating system resource problem, for example, swap space, lack of threads, and so on.

command exited with signal 5, core=false

This error message can occur if you change any default or validation policy, and the script has an error. For example, you might want to change the **tl_def_dist_mode** to **LOCAL**. If you re-create the script with the **echo** command, you will have this error because the **echo** command adds a new line character. Therefore, you have to use the **printf** command as follows:

```
#!/bin/sh
printf LOCAL
exit 0
```

'open' failed with code '13': Permission denied

This error occurs in tasks when the user ID running the task does not have permission to write to the log file. If you are receiving this error message on tasks, have the output display on the Tivoli desktop. If the desktop output is correct, but you are receiving code 13, check permissions. On jobs, this error can be seen under two circumstances. Either you do not have permission to write to the log file, or the task specified in the job does not have a valid user ID on the target system.

(14): no permission for 'TaskLibrary/jacquestask' for operation 'run_task'

The person running the job or task does not have the authority to run the task. Check the task access control list (ACL) with the **wgettask** command to see what roles are required to run the task. Check the Tivoli region roles for the administrator. Remember, roles are not hierarchical. Add an appropriate role to the administrator or to the task.

'COMMconnect_host' failed with code '79':'rh2900c' or pctmp109 (Endpoint): ipc_create_remote failed: unable to connect to 10.22.33.208+9494: (67) IPC shutdown

The target endpoint daemon was not running on the endpoint.

(4): resource 'jennydate' not found

The task **jennydate** was deleted, but the job still references it. This error was received when running the job that is referencing the **jennydate** task.

execve failed with code (2)

Check to see if CR/LF terminators are in the task script and remove them. This error could also mean that there is an invalid shell on the first line or there is an invalid path to Perl.

execve failed with code (8)

Check to see if **#!/bin/sh** is missing from the first line of the task.

could not encrypt/decrypt data

Check the validity of installation and interregion passwords when running tasks on targets in an interconnected Tivoli region.

date time: 'tap_make_sid_logon_token' failed with code '77'

This indicates there is a problem with the Windows user account. Either the account from which the task is attempting to run does not have `Log on locally` user rights, the account is disabled, the Tivoli remote access account is invalid, or the account does not have `Password Never Expired` set.

date time (13): attribute 'user ID' not permitted to take on value 'root' in region 'region-name' by validation method 'tl_val_set_uid'

With this error, the task library validation policy for this region is restricting the user. To correct this error, edit the **tl_val_set_uid** policy script to recognize the user ID root.

Scheduler

The scheduler maintains a record of all scheduled jobs. This record tracks scheduled system management tasks over long periods of time. As jobs are run and completed, they are removed from the scheduler queue. Jobs with a repeating schedule are deleted when the repeat cycle is completed.

Problems can occur when an administrator who has scheduled a repeating job leaves the company or moves to another position. In such cases, the person is deleted as a valid Tivoli administrator, but the scheduler still contains jobs scheduled to run in the future that are associated with the Tivoli administrator who no longer exists. In such situations, you need to remove the job and, in most cases, reschedule the job under a different administrator.

Notices

Almost every systems management action in a Tivoli environment results in a notice being posted in a notice group. This is useful for an administrator overseeing the work of other administrators. This section describes some useful actions you might need to perform when troubleshooting notices.

Rereading Old Notices

Suppose you want to look at the old notices in the Authorization group for the root administrator. If you left at least one notice marked as unread, this would work. Yet, the last time you went into the group, you selected all of them as read. Now, when you try to select Tivoli Authorization Notices, you get a `The notice group contains no unread notices` error message. To read previously read notices, you can do one of the following:

- Look at the list of notices from the command line. Use the **wlsnotif** command, but you must know the notice number if you want to view a notice marked as read. The **wlsnotif** command lists only unread notices by default.
- Manually add a new notice to the notice group you want to open using the **wsndnotif** command. Then you can go back to the Tivoli desktop and open the notice group.

The following is a checklist of common problems you can encounter with reading notices:

- Is the appropriate notice group defined to the monitor, distribution file package, and so on?
- Through the command line, can you post to the notice group?
- Are multiple people using a Tivoli desktop? Check the previous notices. Someone else might have read the notice and marked it as read.
- Is the administrator subscribed to the notice group?

Restoring the Notices Database

Sometimes it might be necessary to restore the notices database. A normal restore operation might not work because the backup could be several months old, and the old notices might or might not be relevant. Prior to restoring a previous notices database, especially if you had an unplanned restore of the Tivoli region, explore the information contained in the notice group for any information that might lead to the cause of any problems. For more information, see "Restoring Databases in a Region" on page 41.

Restoring the Bulletin Board Icon to the Desktop

At some point, the bulletin board icon for the notices can be removed from the Tivoli desktop. If you want to add the icon back to the desktop, follow these steps:

1. To find the administrator OID, enter the following command:
   ```
   wls -l /Library/Administrator
   ```
2. Enter the following command:
   ```
   BOID='idlattr -tg $ADMOID bulletin_board Object'
   ```
3. To add the icon back to the specific Tivoli desktop, enter the following command:
   ```
   idlcall $ADMOID add_object $BBOID
   ```

Other useful commands for restoring the bulletin board icon include the following:

wls –l /Administrators/*administrator_name***/Notices**
Returns the notices OID by administrator name

wls –l /Library/BulletinBoard
Returns the list of notices OIDs

wls –ld /Library/TMF_Notice
Returns information about notice collection and the OID

wls –l /Library/TMF_Notice
Returns the list of notice groups

Tivoli desktop

When a Tivoli administrator launches one or more Tivoli desktops, the Tivoli server spawns a process for the desktops of that administrator. Problems can occur if network connectivity is lost between the process and the Tivoli desktop. For example, if a network connection is lost and the administrator exits the Tivoli desktop, the process does not detect the shutdown of the desktop and continues to run. Problems can also occur while using the Tivoli desktop in a firewall environment. For example, if a firewall detects no network traffic between a Tivoli server and a Tivoli desktop within a given timeout period, the firewall might terminate the connection. The process on the Tivoli server does not detect the terminated connection and continues to run on the Tivoli server.

To prevent these situations, you can set the environment variable **DESKTOPS_CHECK_TIMEOUT**. The **DESKTOPS_CHECK_TIMEOUT** environment variable enables the following functions:

- Reverification of existing IOM connections every **DESKTOPS_CHECK_TIMEOUT** seconds.
- Timeout of the initial establishment of IOM connections after **DESKTOPS_CHECK_TIMEOUT** seconds.

Setting **DESKTOPS_CHECK_TIMEOUT** is optional. If **DESKTOPS_CHECK_TIMEOUT** is not set, then there is no timeout.

The **DESKTOPS_CHECK_TIMEOUT** environment variable only applies to the TMR server and is set using the **odadmin environ** command, as shown below:

1. Type the following command:
   ```
   odadmin environ get 1 > temporary_file
   ```
2. Edit *temporary_file* and add the following line:
   ```
   DESKTOPS_CHECK_TIMEOUT=60
   ```

3. Type the following command:
   ```
   odadmin environ set 1 < temporary_file
   ```

After the **DESKTOPS_CHECK_TIMEOUT** environment variable is set, the TMR server object dispatcher service must be stopped and then started. Stop and restart the object dispatcher service on the TMR server using the following command:

```
odadmin reexec 1
```

When setting **DESKTOPS_CHECK_TIMEOUT**, you should consider the number of simultaneously active desktops and the network configuration in your enterprise. Setting the **DESKTOPS_CHECK_TIMEOUT** environment variable results in an increase in network traffic.

Tivoli Desktop for Windows

The following troubleshooting tips should help you solve problems if you are unable to make a connection to a managed node with Tivoli Desktop for Windows:

- Check your host name, user account, and password for spelling or case. These fields are case-sensitive.
- Ensure that TCP/IP is working and able to connect to your host. Use **Telnet** to verify that you are able to connect to and log in to your host using the same user account and password.
- Ensure that the object dispatcher is running on the managed node.
- Ensure that remote connections are allowed to your managed node. For more information, see the *Tivoli Enterprise Installation Guide*.
- To ensure that your login account is a valid Tivoli administrator, follow these steps:
  1. From a working Tivoli desktop, open the Administrators window.
  2. Verify that your login account is one of the listed administrators or run the **wgetadmin** command to view the login name for administrator.
- To ensure that your login account has a valid, nonblank group property, follow these steps:
  1. From a working Tivoli desktop, open the Administrators window.
  2. Right-click the Administrator's icon.
  3. Click **Edit Properties** to bring up the Administrator Properties window. Ensure that the **Group Name** field contains a valid, nonblank value.
  4. Ensure that you are connecting to a system whose managed node label is the same as its host name.

RIM Host Configuration

The following troubleshooting tip should help you if your RIM host is not exiting when connectivity to the client application is lost.

RIM agents communicate with the client application using an IOM channel. The client application sends commands and data to the agent over the IOM channel when it needs to access the database. The RIM agent polls the IOM channel waiting for a command from the client application.

If either end of the IOM channel is destroyed, the entire connection becomes void, and the client application must establish a new connection. Such a case occurs when the client application, such as Tivoli Enterprise Console, loses network connectivity. While the client application establishes a new connection with a new RIM agent, the original RIM agent is unaware that the other end of the IOM channel has been destroyed and continues to wait for the client application to send a request. The main problem with this scenario is that the original RIM agent still maintains a connection to the database. Over time, if this scenario occurs multiple times, these orphaned agents build up and use available database connections. The system administrator must kill these RIM agents in order to release the database connections.

In order to prevent this problem, a timer can be set on the RIM agent. If the timer is set and the timeout period expires without the RIM agent receiving communication from the client application, then the RIM agent throws an exception and exits. This occurs only if the timer is active and the RIM agent does not receive communication from the client application for the duration of the timeout period. By default, this timer is off and the RIM agent waits indefinitely for the next command from the client application.

The timer can be activated and configured by using the **odadmin** environment variable ***RIM_IOM_TIMEOUT***. The value of this variable is the maximum amount of time in hours that the agent waits to receive a command from the client. If the timeout value is set to 0 or less, then the agent does not use a timer and waits indefinitely. If the timeout value is set to a positive, non-zero integer, then the agent waits for that period of time. The maximum value is 48.

System variable	Value	Is the timer active?	Agent behavior
RIM_IOM_TIMEOUT	Not set	No	The agent waits indefinitely for the next command.
	≤ 0	No	The agent waits indefinitely for the next command.
	≥ 1	Yes	The agent waits up to the number of hours specified for the next command. The maximum number of hours that the agent waits is 48. If the number of hours specified is greater than 48, the agent waits up to 48 hours.

The following setting causes the RIM agent to exit after 3 hours if the client application has not sent any communication.

```
RIM_IOM_TIMEOUT=3
```

Using Tivoli Enterprise across Firewalls

When using Tivoli Enterprise in system environments with firewalls, there are several considerations to keep in mind. In addition to the complications arising from firewall restrictions, environments with network address translators (NAT) also can cause unexpected results with Tivoli Management Framework functionality. To minimize these complications and learn how NAT and firewalls are supported within the Tivoli environment, see the *Tivoli Enterprise Across Firewalls* redbook located at:

http://www.redbooks.ibm.com./solutions/tivoli/

Appendix. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents.You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION ″AS IS″ WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Trademarks

Index

Printed in USA

GC32-0807-02