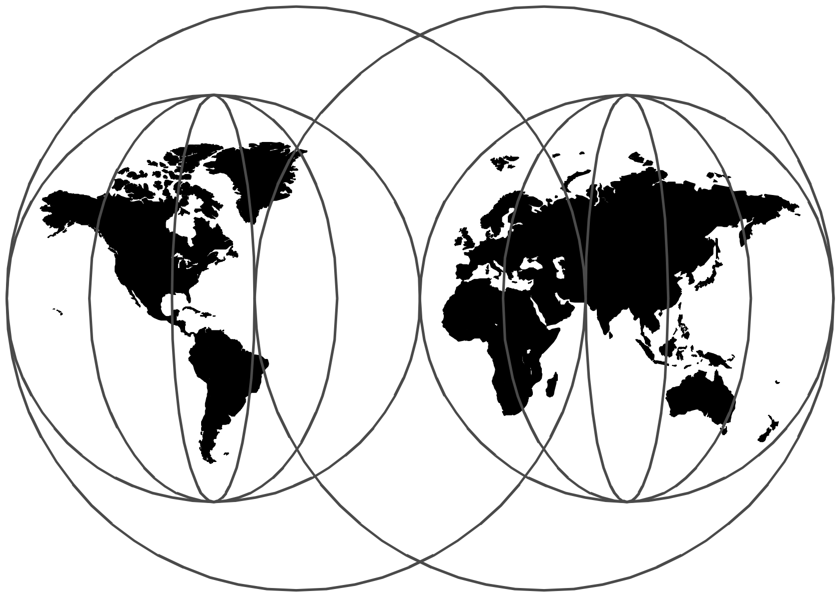
Tivoli

IBM

All About Tivoli Management Agents

Yoichiro Ishii, Hiroshi Kashima



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5134-00

SG24-5134-00

International Technical Support Organization

All About Tivoli Management Agents

March 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix C, “Special Notices” on page 407.

First Edition (March 1999)

This edition applies to the Tivoli program product, "Tivoli Management Framework" Version 3.6 for use with AIX, Windows NT, Windows 95 and Windows 98.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. OSJB Building 045 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

Contents

Figures

Tables

Preface

The Tivoli Management Agent is delivered in a variety of ways, from preloaded systems, to being packaged with communications adapters and operating systems. When a company decides to deploy the Tivoli Framework, how can they take advantage of the Tivoli Management Agents already existing on many of their users' systems? This redbook investigates the prepackaging of Tivoli Management Agents and documents techniques to quickly enable these agents and to immediately utilize them in conjunction with the Tivoli Management Applications.

We will investigate and document how to start the agents, how to have them automatically insert themselves into a Tivoli Management Region, and how to have the systems on which they execute added to the appropriate application's profile managers. This book will help customers, and those who provide services for customers, build a plan for the rapid deployment of Tivoli in environments ranging from small to large.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Yoichiro Ishii is an Advisory I/T Specialist at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM classes worldwide on all areas of System Management and Network Management. Before joining the ITSO in mid-1998, Yoichiro worked in the technical support department in IBM Japan and supported large government projects from 1995 to 1998.

Hiroshi Kashima is an Advisory Systems Engineering Specialist in IBM Japan. He has many years of experience in Networks and Systems Management, and in Banking Application Development using Java, C++, and CORBA. His areas of expertise include distributed programming on the AIX and Windows NT environment. He has written extensively on installation, configuration, and application programming.

Thanks to the following people for their invaluable contributions to this project:

Bart Jacob
International Technical Support Organization, Austin Center

Tara Campbell
International Technical Support Organization, Austin Center

Marcus Brewer
International Technical Support Organization, Austin Center

Russell Hill
Tivoli Systems

Victoria Stevens
Tivoli Systems

Rich LaSota
Tivoli Systems

Gene Cherry
Tivoli Systems

Martin Voshell
Tivoli Systems

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 423 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

For Internet users	`http://www.redbooks.ibm.com`
For IBM Intranet users	`http://w3.itso.ibm.com`

- Send us a note at the following address:

  `redbook@us.ibm.com`

Chapter 1. Introduction

The Tivoli Management Framework provides a set of common services and facilities that enable powerful systems management applications. This framework provides benefits to developers who want to take advantage of services and facilities that hide the complexity of the networking environment. By doing so, the Tivoli Management Framework allows the developer to concentrate on developing solutions that apply across a wide range of operating environments.

Likewise, the framework is valuable to those responsible for managing complex environments because it provides common user interface elements and hides differences in the operating environments of managed systems.

The Tivoli Framework is based on industry standards, such as the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA), and has had wide acceptance with a large number of system management application developers. Designing a framework, such as the Tivoli Management Framework, requires meeting two (sometimes conflicting) criteria: stability and extensibility.

Stability provides application developers with confidence that the applications they develop will continue to run when new versions of the framework become available. Extensibility provides customers with the knowledge that the framework can evolve over time to meet their changing requirements.

Version 3.2 of the Tivoli Management Framework introduced major new extensions to the framework's architecture. These new extensions included the Lightweight Client Framework (LCF) architecture. This version of the framework is a testament to both the stability and extensibility of the Tivoli product's architecture. Although the framework at Version 3.2 supported these extensions, few of the Tivoli Management Applications took advantage of them until their next release, which was at Version 3.6.

Now that Tivoli 3.6 has been released, and the applications take full advantage of the new architecture extensions, we have an even more powerful set of functions and services for distributed systems management. Version 3.6 of the Tivoli Management Framework and applications can be installed and configured in the same way as previous versions. In addition, applications can now take advantage of new client types that run a component called the Tivoli Management Agent (TMA). Utilizing the TMA with Version 3.6 provides a new level of extensibility and allows you to scale your management across the entire enterprise.

In parallel with the new extensions for Tivoli, Tivoli has announced a Tivoli Ready logo. This logo on a partner company's product indicates that the product has passed rigorous product certification testing by Tivoli. The Tivoli Management Agent will be provided with and used by Tivoli Ready products. The management interface provided by the TMA is the preferred management interface to be used by Tivoli-based management applications.

Therefore, Tivoli Ready products are truly ready to take advantage of Tivoli's technologies and to extend the management services of Tivoli software to manage applications and devices. In Chapter 2, "Tivoli Ready with Tivoli Management Agent" on page 31, we will talk about Tivoli Ready products in more detail.

This redbook describes the features and services provided by the TMA in detail. Solutions and examples of using the TMA are also provided in this redbook.

1.1 Tivoli 3.6

Version 3.6 of the Tivoli Enterprise products are an extremely strategic set of products. From now on, we will refer to this set of products as Tivoli 3.6. Tivoli 3.6 encompasses almost all features and services that were provided by the previous version of the Tivoli products and also extends these features. Tivoli 3.6 provides real extensibility and flexibility for customers.

As we mentioned, LCF architecture was available in Version 3.2 of the Tivoli Management Framework. However, there were few applications that could support the LCF Endpoint. In Tivoli 3.6, all Tivoli Management core applications support the LCF architecture and can run on TMA machines.

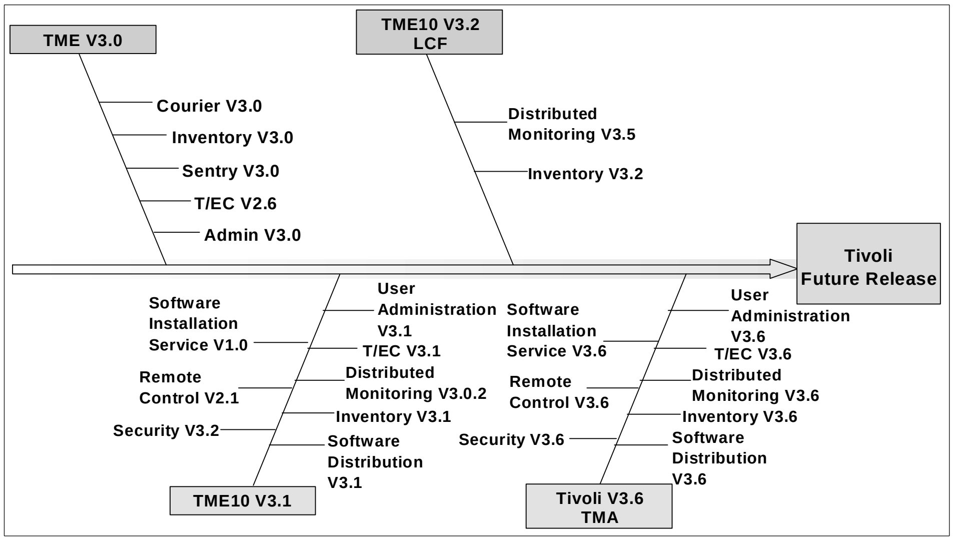


Figure 1. Tivoli Products History

1.1.1 Advantages of Tivoli 3.6

The most significant enhancements across the Tivoli 3.6 product set include:

- Providing TMA support for applications
- Support for additional platforms
- Internationalization

We discuss the details of the TMA throughout the rest of this redbook. Therefore, this section mainly introduces the other enhancements of Tivoli 3.6.

1.1.1.1 Supporting Many Platforms

Tivoli 3.6 supports many platforms with the TMA. The following are the main platforms Tivoli 3.6 TMA supports:

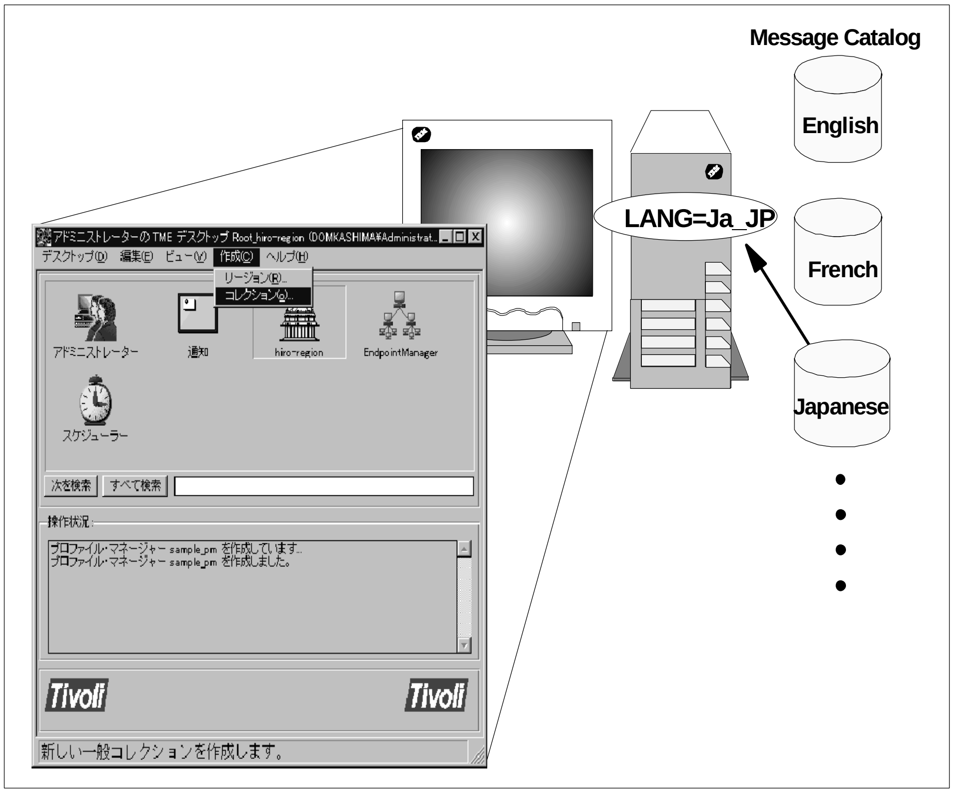
- UNIX
- Windows NT
- Windows 98, 95, 3.x
- Netware 3, 4
- OS/2
- AS/400

- OS/390

As you can see, Tivoli 3.6 supports many platforms—from the PC to the mainframe. Once the Tivoli Framework is installed, the customer can manage all system types running the TMA or other Tivoli client software with a single operation, using the Desktop interface and the CLI. This is because the framework and applications provide platform independence. This means the type of target on which a management operation is to be performed is transparent to the administrator or program initiating the operation. Therefore, seamless operations become available for multiple platforms in the Tivoli 3.6 environment. This platform independence is one of the many benefits provided by the Tivoli architecture.

1.1.1.2 Internationalization

National language support is one of the most important features for customers who don’t use English, because the user interface is very important in systems management software. Tivoli 3.6 provides the internationalization feature using the implementation illustrated in Figure 2 on page 5.



```
プロファイル・マネージャー sample_pm を作成しています...
プロファイル・マネージャー sample_pm を作成しました。
```

新しい一般コレクションを作成します。

Figure 2. Internationalization Implementation of Tivoli 3.6

Previous versions of Tivoli had separate source code for each supported non-English language. In the internationalization implementation, the Tivoli products contained hard-coded message catalogs in their source files. Keeping the source code across all languages at the same level was difficult, at best. As a result, patch modules were needed for each language’s version. When these versions were not kept synchronized, it caused a lot of confusion for the customer.

The internationalized Tivoli 3.6 has one common set of source code with many language code sets. This means that supporting other languages is easier and faster than with previous versions of Tivoli. This implementation of the internationalization feature is similar to the implementation of the IBM AIX operating system. For example, Figure 2 shows how to load the Japanese language code set. In this case, we simply set the system language environment to Japanese, and Tivoli 3.6 displays the Desktop with Japanese characters. All messages are built into message catalogs that are maintained separately from the source code.

Tivoli 3.6 presently supports the following languages:

- Chinese
- English
- French
- Japanese
- Korean
- Portuguese

1.1.2 Co-Existence of Different Managed Resources

Although the Tivoli Management Agent provides a powerful interface for managing a wide variety of systems, Tivoli 3.6 still supports the following types of managed resources used in previous versions, and also keeps consistency among these different managed resources:

- Managed Node
- PC Managed Node
- Netware Managed Site

It is possible to simply upgrade to Tivoli 3.6 and maintain your current management architecture using the previous managed resource types. To take advantage of the TMA in an existing environment that is being upgraded to Version 3.6, Tivoli provides the Tivoli Migration Toolkit.

This toolkit helps with the creation of managed resources based on the Tivoli Management Agent (often referred to as Endpoints) and the migration of profiles and other objects related to the management applications, so they can take advantage of the new client type.

The TMA replaces the above managed resources, while using a surprisingly small amount of disk space and memory. As we mentioned, Tivoli 3.6 provides both stability and extensibility to all customers. This redbook is designed to help Tivoli 3.6 users build a strong understanding of the TMA and its new architecture.

1.2 Overview of Tivoli Management Agent

The most visible new feature of Version 3.6 of the Tivoli Management Framework is the Tivoli Management Agent (TMA), previously called the Lightweight Client Framework (LCF) Endpoint. The TMA is an extension of the classic TME 10 Framework that increases scalability of TMRs, while reducing the hardware and software requirements on the managed systems. The following sections describe this new architecture and its main components, including the TMA.

1.2.1 TMA Introduction

The TMA-related extensions to the framework introduce three object types that represent system roles in a TMR:

- Endpoint (TMA)
- Endpoint gateway
- Endpoint Manager

Although each of the above items logically represents a different system's role in the Tivoli environment, it should be noted that a single physical system can contain more than one of the above object types. That is, one system could contain an Endpoint Manager, an Endpoint gateway and an Endpoint. (However, in most environments, Endpoint gateways will reside on different systems than Endpoint Managers.)

1.2.1.1 Endpoint

The Endpoint is installed on systems to be managed. The Endpoint does not include any capability to perform management operations on other systems. That is, like most end-user workstations, these systems will be managed, but they will not be involved in the management of other nodes. More specifically, the Endpoint does not provide true Tivoli Desktop or command line interface so these resources in the network can not be managed from the Endpoint.

The Endpoint function resides in the node to be managed. It runs as a small daemon, or background task. This daemon is called the `lcfd`. It is responsible for executing methods at the request of a managing system. Its only connection to and knowledge of the rest of the Tivoli world is through an Endpoint gateway.

When an Endpoint is installed, a minimal number of files are installed on the managed system. Functionally, the only thing that is installed is the `lcfd` itself. When an application invokes a method to be executed on the managed system (Endpoint), the method is automatically downloaded to the Endpoint and executed by the `lcfd`. The methods that are downloaded to the Endpoint are cached at the Endpoint. As long as that method stays in the cache, it does not need to be downloaded again at a second invocation of the same method. The cache on the Endpoint is a disk cache. Therefore, it is persistent across IPLs of the managed system.

1.2.1.2 Endpoint Gateway

The Endpoint gateway is a software component that runs on a full Tivoli Managed Node, enabling the Managed Node to operate as a gateway between a cluster of Endpoints and the rest of the TMR. Each TMR can have multiple Endpoint gateways. The number of gateways will depend on factors such as available system resources, the number of Endpoints, and network topology. Currently, one TMR Server can handle up to approximately 200 Endpoint gateways. This limit is actually based on the number of Managed Nodes that one TMR Server can manage. There is no precise limit to how many Endpoints one Endpoint gateway can handle. This will depend on system resources, performance requirements and the type of management being performed. However, testing has been done that indicates that in many environments, up to 2000 Endpoints or more may be supported by a single Endpoint gateway.

The Endpoint gateway performs the following functions:

- Listens for Endpoint login requests

  The Endpoint gateway maintains (with help from the Endpoint Manager) a list of the Endpoints that it is responsible for. As the Endpoints come online, they will attempt to login to a specific Endpoint gateway or broadcast a message searching for an Endpoint gateway. The Endpoint gateway will receive these transmissions, and if responsible for the given Endpoint, will proceed with the login process. If the entry of the Endpoint does not exist in the Endpoint gateway's list, the Endpoint gateway will forward the login request to the Endpoint Manager so that an Endpoint gateway can be assigned to the Endpoint. This Endpoint login procedure is called initial login. We explain the Endpoint login procedure in detail in Chapter 5, "Anatomy of TMA Behavior" on page 135.

- Listens for downcall method requests

  Method invocations from other nodes that are targeted as one of the Endpoints a Endpoint gateway is responsible for will pass through the Endpoint gateway. For downcalls, the Endpoint gateway is transparent. When it receives a method invocation targeted for the Endpoint for which it is the Endpoint gateway, it will pass the method invocation (along with the method and any dependencies, if necessary) on to the Endpoint. It will then wait for any method results and pass them back to the original caller.

- Listens for Endpoint upcall requests

  If the Endpoint needs to invoke an operation on another system, it must invoke a method on its own Endpoint gateway. The appropriate application that is stored in the Endpoint gateway will supply the method. This method will then take advantage of the full function of the Managed Node on which it resides to resolve the location of the target object and invoke the appropriate method(s) upon it.

- MDist Repeater activities

  The Endpoint gateways are automatically defined as MDist (multiplex distribution) repeaters for all of the Endpoints they serve. In the traditional Tivoli Framework, we defined MDist repeaters using the `wrpt` command. The MDist repeater function provides the fan out facility for the distribution of files and data in the Tivoli environment. Therefore, if the same file is being distributed to a set of Endpoints using the same Endpoint gateway, the file only needs to be sent once to the Endpoint gateway, and the Endpoint gateway will then handle distributing the file to the individual Endpoints. This gives you the benefit of an intelligent distribution mechanism with little or no administrative overhead.

1.2.1.3 Endpoint Manager

The Endpoint Manager stores the association between the Endpoint gateways and Endpoints. Specifically, it performs the following functions:

- The Endpoint Manager maintains the Endpoint list which keeps track of every Endpoint in the TMR. This list tracks which Endpoint gateway is responsible for each of the Endpoints. Based on site-specific settings, the Endpoint Manager reassigns Endpoints if the Endpoint gateway is unavailable, and dynamically adds new Endpoints as they appear on the network. The Endpoint list contains the information necessary to uniquely identify and manage the Endpoints. This includes:

Name of the Endpoint	A user-friendly name for use in the Tivoli Name Registry (TNR).
Endpoint’s interpreter	The string denoting the platform and operating system of the Endpoint (such as NT or OS/2).
Object dispatcher identifier (odnum)	A unique system identifier for the Endpoint.

Endpoint Gateway	The name of the Gateway that is responsible for communications with the Endpoint.

• The Endpoint Manager plays a role in enforcing site-specific system policies. For example, policies may be put in place that specify which Endpoint gateway will be assigned to new Endpoints joining the network. These policies could base their decisions on a variety of information regarding the Endpoint, which is included in the Endpoint’s initial login request for a new Endpoint gateway.

1.2.2 Tivoli Management Agent and Tivoli Desktop

In general, during day-to-day activity, the Tivoli administrator will see little difference when managing systems with the TMA as compared to Managed Nodes in previous versions. That is, Tivoli applications will fully support the Endpoint, and you will use the Endpoint as subscribers to profile-based applications just as you used Managed Nodes in previous versions of the Tivoli Management Framework.

However, there are some additions and changes to the Tivoli Desktop that the administrator will notice. For instance, a new icon is added to the Tivoli Desktop to represent the Endpoint Manager. You can create and delete an Endpoint gateway from this icon. You may use this Endpoint Manager resource to view a list of all Endpoint gateways and the Endpoints managed by each Endpoint gateway.

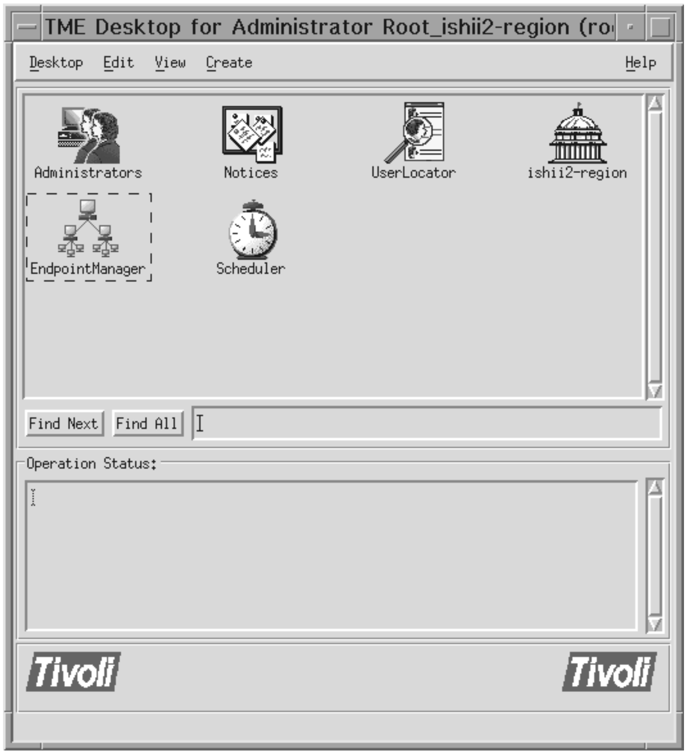


Figure 3. The EndpointManager Icon on the Tivoli Desktop

In addition, in the traditional Tivoli environment, Managed Nodes were displayed as such in the various policy regions. Endpoints do not appear in policy regions by default. The decision to have them not linked into a policy region was based on the desire to keep the performance of the Desktop at a reasonable level, even when thousands of systems are being managed. However, if desired, you may link an Endpoint with a policy region to have its icon displayed.

Although the icons representing the systems do not show up by default within a policy region, Endpoints will be displayed in dialog boxes showing potential subscribers to profiles and jobs. Therefore, you will still be able to use the Desktop GUI to manage systems running the TMA.

1.2.3 Tivoli Management Agent and Command Line Interface

Version 3.6 of the Tivoli Management Framework includes commands specifically related to helping manage Tivoli Management Agents. In this section, we summarize these commands. Please refer to the *Tivoli Framework*

Reference Manual and the *Tivoli Framework Release Notes* for detailed information about these commands.

`winstlcf`	Installs an Endpoint on a UNIX or Windows NT workstation. For more information on installing an Endpoint, please see Chapter 3, “Tivoli Management Agent Installation” on page 41.
`wsetpm`	Enables/disables the profile manager to operate in dataless mode. Since Endpoints don’t include a Tivoli object database, profile information is not stored on managed systems the way it is for full Managed Nodes. The profile managers must be enabled for dataless operation to allow Endpoints as subscribers. In Chapter 6, “TMA and Tivoli Management Applications” on page 211, we will talk about the dataless profile manager in detail.
`lcfd`	Starts the Endpoint daemon (`lcfd`) on the Endpoint and installs or removes the daemon as a service on Windows NT.
`lcfd.sh`	Starts the Endpoint daemon (`lcfd`) on the UNIX Endpoints.
`wcrtgate`	Creates an Endpoint gateway.
`wdelgate`	Deletes an Endpoint gateway.
`wgateway`	Starts, stops and lists the properties of an Endpoint gateway. This command is also used to synchronize the Endpoint gateway method cache with that on the TMR Server.
`wep`	Performs actions on the Endpoint information contained in the Endpoint list maintained by the Endpoint Manager. This command can list or alter the information related to the Endpoints.
`wadminep`	Performs a variety of administrative actions on the Endpoints. In general, once the Endpoints are installed, there is little that needs to be done to administer the `lcfd` daemon. However, this command would be useful when first installing and testing the Endpoint. Please refer to Chapter 7, “Advanced Knowledge of the TMA” on page 251 for more information.
`wgeteppol`	Lists the body and the constant values of Endpoint policy methods. Use this command to extract a current Endpoint policy method, which you can modify and then replace with the `wputeppol` command.
`wputeppol`	Replaces the body of an Endpoint policy method.

We will introduce examples of using most of these commands throughout the rest of this book.

1.2.4 Tivoli Management Agent and Web Interface

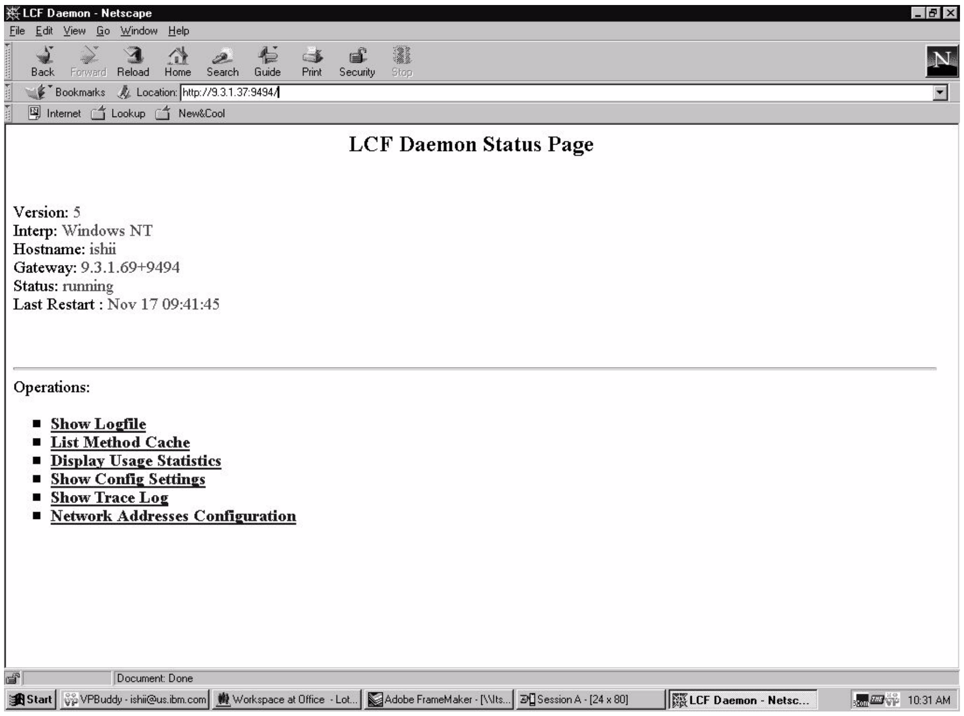
Version 3.6 of the Tivoli Management Framework includes integrated HTTP daemons that allow administrators to perform management operations through a Web browser interface like Netscape Navigator or Microsoft Internet Explorer. These daemons are automatically installed on TMR Servers, Managed Nodes and Endpoints. In this part of the book, we will focus on the Endpoint's Web server function.

1.2.4.1 Accessing the Endpoint Web Interface

To access the HTTP daemon included in the Endpoint, simply use the URL that consists of the system name (or IP address) and the port number that the Endpoint uses for communication with the Endpoint gateway. Normally, the default port number is 9494. Therefore, to access the Endpoint known as `ishii` we use our Web browser to access:

```
http://ishii.itsc.austin.ibm.com:9494
```

Our Web browser now displays the Endpoint's first page, as shown in Figure 4 on page 13.


LCF Daemon - Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://9.3.1.37:9494/
Internet Lookup New&Cool
Document: Done
Start VPBuddy - ishii@us.ibm.com Workspace at Office - Lot... Adobe FrameMaker - [\\Its... Session A - [24 x 80] LCF Daemon - Netsc... 10:31 AM

LCF Daemon Status Page

Version: 5
Interp: Windows NT
Hostname: ishii
Gateway: 9.3.1.69+9494
Status: running
Last Restart : Nov 17 09:41:45

Operations:

- Show Logfile
- List Method Cache
- Display Usage Statistics
- Show Config Settings
- Show Trace Log
- Network Addresses Configuration

Figure 4. The Endpoint Web Interface

1.2.5 Functions Provided through the Endpoint Web Interface

The Endpoint's Web server provides very specific information and services to administrators. The administrator can use a Web browser to query information about the Endpoint, as well as use it to change its configuration parameters. There are seven specific pages that display information and allow an administrator to alter the Endpoint's configuration parameters. The following describes the information and operations provided by each page.

LCF Daemon Status Page:

This is the primary page on the Endpoint Web interface. The top half of this page provides information regarding the Endpoint, including:

- **Version** - The version of the lcfd on the Endpoint
- **Interp** - The operating system on the Endpoint
- **Hostname** - The Endpoint's host name
- **Gateway** - The Endpoint gateway's address and port number
- **Status** - The current status of the Endpoint
- **Last Restart** - The date and time that this Endpoint was last started

Most of this information can also be accessed with the `wadminep` command.

Show Logfile Page:

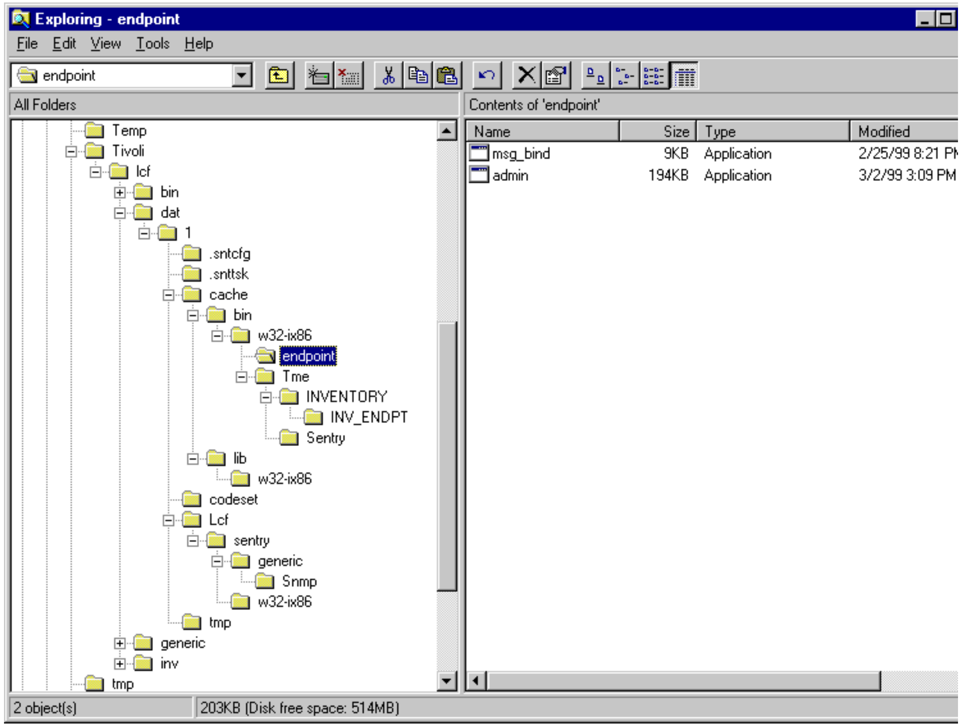
This page simply displays the contents of the lcfd.log file. Therefore, it is the same as browsing the lcfd.log file from command line interface. Please refer to Chapter 7, "Advanced Knowledge of the TMA" on page 251 for more information about the lcfd.log file.

List Method Cache Page:

It is sometimes useful to be able to see a listing of the methods that are currently available in the Endpoint method cache. We can get the same information using the `wadminep endpoint_label view_cache_index` command as follows.

```
# wadminep ausres12 view_cache_index
Performing browse mode 'view_cache_index' on endpoint 'ausres12'
Index|Version|Size|Flags|OOC|Hits|TimeLastHit|TimeRead|Prefix|Path|
0|0x35cfa903|9216|00000000|False|1|19 Feb 1999 17:15:24|19 Feb 1999 17:15:24|C:\
Program Files\Tivoli\lcf\dat\1\cache|\bin\w32-ix86\endpoint\msg_bind|
1|0x35b3d8fa|1671|0x000008|True|1|19 Feb 1999 17:15:24|19 Feb 1999 17:15:24|C:\P
rogram Files\Tivoli\lcf\generic\msg_cat\C|\msg_cat\C\GatewayCatalog.cat|
2|0x35b3d91b|1900|0x000008|True|1|19 Feb 1999 17:15:24|19 Feb 1999 17:15:24|C:\P
rogram Files\Tivoli\lcf\generic\msg_cat\fr_FR|\msg_cat\fr_FR\GatewayCatalog.cat|
3|0x35b3d905|2252|0x000008|True|1|19 Feb 1999 17:15:24|19 Feb 1999 17:15:24|C:\P
rogram Files\Tivoli\lcf\generic\msg_cat\ja_JP|\msg_cat\ja_JP\GatewayCatalog.cat|
4|0x35b3d910|1831|0x000008|True|1|19 Feb 1999 17:15:24|19 Feb 1999 17:15:24|C:\P
rogram Files\Tivoli\lcf\generic\msg_cat\pt_BR|\msg_cat\pt_BR\GatewayCatalog.cat|
5|0x35cfa903|194560|00000000|False|1|19 Feb 1999 20:18:59|19 Feb 1999 20:18:59|C
:\Program Files\Tivoli\lcf\dat\1\cache|\bin\w32-ix86\endpoint\admin|

#
```

The methods shown on this page would be stored in the cache directories on the Endpoint. We can confirm this using a tool to browse the directory tree, such as Windows Explorer, as shown in Figure 5 on page 15.



Name	Size	Type	Modified
msg_bind	9KB	Application	2/25/99 8:21 PM
admin	194KB	Application	3/2/99 3:09 PM

2 object(s) | 203KB (Disk free space: 514MB)

Figure 5. The Methods Stored in the Cache on TMA

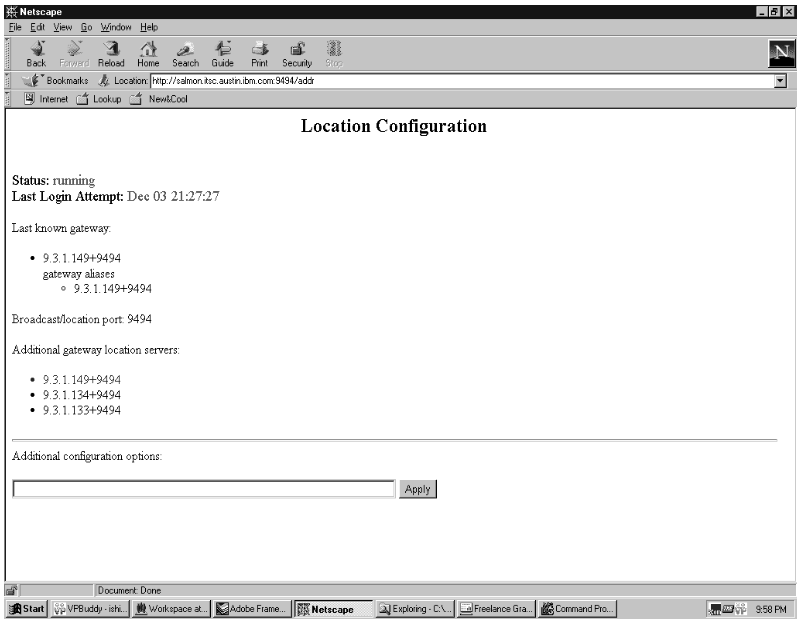
Display Usage Statistics Page:

This page, as its name implies, provides statistics related to the number of downcalls that have been issued and the hit and miss rate for our method cache. We can obtain more detailed information regarding the downcall, downcall history, cache, and so forth, using the `wadminep endpoint_label view_statistics` command.

Show Config Setting Page:
This page simply displays the current configuration setting for the Endpoint. The information presented is made up of information contained in the last.cfg file and lcf.dat file. More information on these files is presented in Chapter 4, “Configuring the TMA Environment” on page 105. We can receive similar information to that presented on this page by using the `wadminep endpoint_label view_config_info` command.

Show Trace Log Page:
This page allows the administrator to view the trace log of messages sent and received at the Endpoint.

Network Address Configuration Page:
The Network Address Configuration page displays information about the current Endpoint settings, but more importantly, it allows the administrator to change the configuration of the Endpoint. In the dialog presented for additional configuration options, you can specify any parameters supported by the `lcfd` command. See the *Tivoli Framework Reference Manual* for more detailed information on this command and refer to the Chapter 5, “Anatomy of TMA Behavior” on page 135 for more information about the Endpoint Web interface.


Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://salmon.itsc.austin.ibm.com:9494/addr
Internet Lookup New&Cool
Apply
Document: Done
Start VPBuddy - ishi... Workspace at... Adobe Frame... Netscape Exploring - C:\... Freelance Gra... Command Pro... 9:58 PM

Location Configuration

Status: running
Last Login Attempt: Dec 03 21:27:27

Last known gateway:

- 9.3.1.149+9494
  gateway aliases
  - 9.3.1.149+9494

Broadcast/location port: 9494

Additional gateway location servers:

- 9.3.1.149+9494
- 9.3.1.134+9494
- 9.3.1.133+9494

Additional configuration options:

Figure 6. The Network Address Configuration Page

For instance, you can use this page to alter the default Endpoint gateway for the Endpoint. Once you have entered the configuration options and selected **Apply**, you will be prompted for a user ID and password. To obtain the proper http password, the TMR Administrator (who needs senior or super authority) can issue the `wep` command as follows:

```
# wep ishii get httpd
tivoli:rT!*‘un
```

This will display the current user ID and password for the specified Endpoint. The default user ID is `tivoli`. The administrator may also change the current user ID and password by issuing the following command:

```
# wep ishii set httpd userid:password
```

To go back to changing the configuration via this Web page, once the user ID and password are properly entered, the configuration changes are applied to the Endpoint and the Endpoint is restarted with the new configuration.

Note that, as described in detail later, changing the default Endpoint gateway for an Endpoint through the above mechanism does not change policy enforced by the Endpoint Manager. This only identifies the Gateway the Endpoint will initially contact to log into the TMR. The Endpoint Manager will ultimately determine which Gateway will take ownership of the Endpoint. Making this change through the Web interface is one way of forcing the Endpoint to perform a new initial login sequence to obtain its Gateway if its primary Gateway is unavailable.

1.3 Advantages of the Tivoli Management Agent

In general, the TMA can replace the full Managed Node and provide the same level of manageability to the managed system with far less resource utilization. However, there are still some differences for some operations, as shown in the following comparison table.

Note

In this table, we assume a Windows NT Managed Node. The required resources might be somewhat different on other platforms. Please refer to the latest *Framework Release Notes* for detailed information.

Table 1. Comparison of the Features of the TMA and the Full Managed Node

Features	**Tivoli Management Agent**	**Full Managed Node**
Disk	Approximately 1 MB [1]	Approximately 100 MB
Memory	Approximately 1 MB	Minimum 24 MB
Nodes per TMR	Unlimited	200 Nodes
Load of TMR Server	Off-loaded	Heavy
Object DB	None	Exist
Tivoli Applications	Core Applications Only	Full Support
CLI	Not Available	Full Support
MDist	Support	Support
CCMS [2]	Dataless Profile Manager	Profile Manager
Web Interface	Support	Not Available
Version Upgrade	Easy	Complicated
Management Topology	Three-Tiered Structure	Two-Tiered Structure

Features	Tivoli Management Agent	Full Managed Node
Server and Client	Configurable (Endpoint Login)	Fixed
Preloaded Module	Support	Not Available
T/EC, RIM, and so forth [3]	Not Available	Full Support

1. By default, the TMA could use up to 20 MB disk space for method cache.
2. Configuration and Change Management System.
3. TMR Server, Endpoint gateway, MDist Repeater, T/EC Server, Software Distribution Filepack Source System, Systems hosting Tivoli Plus Modules, Systems using CLI Interface.

1.3.1 Less Disk and Memory Utilization

As you can see, the lcfd daemon itself uses only 1 MB of memory and 1 MB of disk space. Methods which are dynamically downloaded as needed are in addition to this. However, they are usually relatively small and in the range of 300 KB. On the other hand, the full Managed Node used more resources, and this additional resource was often a concern when managing end user systems. In an ideal world, the resources required to manage a system should be minimal. This difference is one reason the TMA is desirable.

1.3.2 Increased Scalability with Reduced Complexity

Due to the architecture of Managed Nodes and the distributed database component that they control, a single TMR is typically limited to around 200 Managed Nodes. For environments with more than 200 systems to manage, the solution was provided through the capability to interconnect TMRs and manage the resources in one TMR from the other. This solution provided many benefits, but also came at a cost of increased complexity.

With the new architecture, a single TMR can now support many thousand Endpoints. Therefore, there will be less of a requirement to interconnect TMRs, which will simplify your Tivoli deployment. If you have interconnected TMRs for other reasons, such as geography or organizational requirements, this is fine. Endpoints can be managed across interconnected TMRs as well. Please refer to Chapter 5, “Anatomy of TMA Behavior” on page 135 for more information about interconnected TMRs.

1.3.3 Lighten the Load on your TMR Server

In Version 3.6 of Tivoli, a single TMR Server can manage thousands of Endpoints. As a result, one might be concerned about the load on the TMR Server. However, the new architecture is designed to allow Endpoint gateways to off-load many of the functions formerly performed by the TMR

Server. Therefore, although we have a large increase in the number of managed systems, the load on the TMR Server may actually decline.

On the other hand, based on the type of management you are performing, you will want to ensure that your workload is balanced across the Endpoint gateways in your environment.

1.3.4 Simplifying the Tivoli Object Database

One of the core components of the Tivoli Framework is its distributed object database. This database is controlled by the TMR Server, but has components on every Managed Node in the TMR. Most management operations result in information being written to or accessed from this distributed database. One of the key limiting factors to the size of a TMR is the amount of resource it takes to keep this database synchronized and performing well. In some cases, a database check or backup (using the `wchkdb` or `wbkupdb` commands) could take several hours or more.

This database on each Managed Node could also require a large amount of disk space. On a managed system, we do not need to consume large amounts of resources solely for systems management.

One of the key aspects and advantages of the new architecture is that the TMA does not maintain an object database. This keeps the amount of resource required on the managed system to a minimum, while simplifying the management of the object database that is still maintained on the TMR Server and any remaining Managed Nodes.

In an ideal environment, we would have fewer Managed Nodes, but many more managed systems through the TMA. With fewer Managed Nodes, the maintenance required on the object database will be reduced, increasing performance and reliability. In addition, we achieve these benefits without losing application functionality.

1.3.5 Support of Tivoli Applications

Although all of the core Tivoli Management Applications now support TMA, some application functions, and other applications such as Tivoli Management Modules, still require the function of a Managed Node. However, over time there will be less and less of a requirement to retain Managed Nodes in your environment as more applications provide full support for the TMA.

1.3.6 Command Line Interface

Another distinguishing difference between Managed Nodes and Endpoints is the availability of a command line interface. There are a large number of Tivoli commands to perform management operations from a command line. These commands allow an administrator to perform complex management operations and to perform almost anything that could be done via the Desktop GUI. (Of course, the user of the command line must be a Tivoli administrator with the proper authority.)

These commands are available on every Managed Node. Even if a Managed Node were an application server or end user system, the commands are still installed. Thus, any Managed Node has the potential to become a management station, which is a workstation from which management operations are invoked.

The Endpoints was designed to be as small as possible. It was designed to be a managed system, and not a management workstation. Therefore, the vast majority of Tivoli commands are not available on Endpoints. For this reason, some systems frequently accessed by Tivoli administrators may be best installed as Managed Nodes to provide the management workstation capability.

This brings up another important point. Endpoints can peacefully coexist with a Managed Node on the same system. That is, you could install a Managed Node on a system to allow it to become a management workstation and, at the same time, install an Endpoint on it for managing that system.

1.3.7 MDist Repeater Function

The MDist repeater function is one of the most powerful functions of the Tivoli Management environment. The MDist repeater function is provided by the Tivoli Management Framework. In the TMA environment, the Endpoint gateway is configured as an MDist repeater automatically when you create the Endpoint gateway. This repeater will serve any and all Endpoints attached to that gateway.

In the full Managed Node environment, you need to configure MDist repeaters using the `wrpt` command. For the administrator, this is one of the most important things to do, because most data that are transferred among the managed resources are sent through an MDist repeater. The placement and tuning of MDist repeaters is important to the overall performance and throughput of the TMR.

1.3.8 CCMS

Most Tivoli applications work through objects called profiles. Information in the profile defines the management action to be taken on a managed system. Managing actions are invoked by distributing these profiles. Profiles and their subscribers (systems that may receive a profile) are managed by another Tivoli object called a profile manager. The CCMS subsystem provides the framework for the operation of profile managers and profiles.

In the traditional framework, profiles were distributed to Managed Nodes and written to the object database on the Managed Nodes. However, since Endpoints do not have an object database, operations of the profile distribution mechanism had to change.

To support the Endpoint that does not have a Tivoli database, CCMS provides a new profile manager type called a dataless profile manager. In the dataless profile manager, creating, configuring, and distributing the profile is absolutely the same as with a normal profile manager, but the main difference is that the information in the profile would not be written to the database. This is even true when distributing to a full Managed Node through a dataless profile manager. We provide more detail about the dataless profile manager in Chapter 6, “TMA and Tivoli Management Applications” on page 211.

1.3.9 Endpoint Web Interface

The Endpoint Web interface is a tool for configuring and controlling the Endpoint. It can be a very useful tool for accessing the Endpoint and potentially fixing problems. This tool is very valuable in cases where one or more Gateways are unavailable and the Endpoint becomes isolated. The Endpoint Web interface uses a different authentication mechanism from other Tivoli clients. That is, other management operations use the Tivoli administrator roles to determine authority. The Endpoint Web interface is dependent on a separate user ID and password as discussed earlier. Only authorized Tivoli administrators have access to this user ID and password, via the CLI. Please refer to the Chapter 5, “Anatomy of TMA Behavior” on page 135 for more information.

1.3.10 Simplified Version Upgrade Operations

An important benefit of the TMA environment is the simplicity of upgrading the level of Tivoli code running on managed systems. For traditional Managed Nodes, the administrator must plan for and explicitly install patches or new versions of Tivoli software. Some of this software can be quite large, and in environments with many Managed Nodes and lower speed communications, can take considerable time. Keeping multiple systems at the same level of software can be a challenge.

However, in the TMA environment, the upgrade process has been greatly simplified for the following reasons:

- The TMA uses software installed on the Endpoint gateway. This means that updated software targeted for Endpoints need only be installed on Endpoint gateways. The number of Endpoint gateways will typically be a very manageable number compared to hundreds or thousands of potential Managed Nodes.
- The auto upgrade function. Once update software is installed on an Endpoint gateway, it is automatically downloaded and installed on the Endpoint the next time the Endpoint uses that particular method. Methods are typically small and, therefore, should cause no significant overhead or delay. In addition to application methods, the`lcfd` daemon itself can be automatically downloaded and upgraded.

Figure 7 on page 24 shows an overview of the how to upgrade software in a TMA environment.

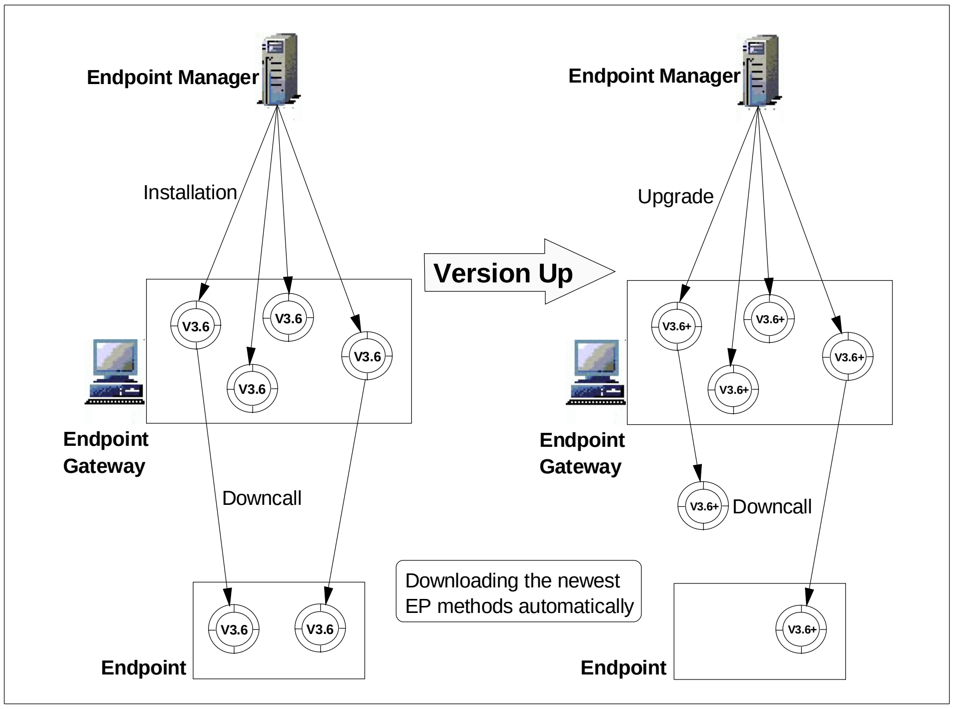


Figure 7. Software Version Upgrade in a TMA Environment

As you can see, to upgrade the version of the software, you simply upgrade the software on the Endpoint gateways. Then, the Endpoint attempts to download the latest software (methods) with the downcall when the Endpoint detects that the available version is greater than the current version of the Endpoint method. In the TMA environment, this is not only for software installation, but also for patch installation.

In the full Managed Node environment, of course, you have to install the software or patches to all Managed Nodes which are managed by the TMR Server. However, the Endpoint only uses software that has been installed on the Endpoint gateway.

The TMA auto-upgrade function enables the `lcfd` daemon to upgrade itself automatically. The auto-upgrade function upgrades the Endpoint software (`lcfd`) if the available version in the Endpoint gateway is greater than the current version of the `lcfd` daemon. We will talk about the auto upgrade in detail later, so please refer to Chapter 5, “Anatomy of TMA Behavior” on page 135 for more information.

1.3.11 Three-Tiered Structure Improves Performance and Availability

The new architecture supporting the TMA has changed the management topology of a TMR from a two-tiered structure to a three-tiered structure. This is a substantial change, and the three-tiered structure provides new and improved functions, performance and availability considerations. We will talk about the TMA management topology in the next section.

1.3.12 Increased Reliability through Endpoint Login Flexibility

In the TMA environment, the relationship between the Endpoint and its Endpoint gateway is highly configurable. To control how the Endpoint connects to the TMR and to which Endpoint gateway it will belong, you can configure the Endpoint login using the following methods:

- Using the Endpoint policy
- Using the options for the `lcfd` daemon
- Using the Endpoint Web interface

This flexibility means that if the assigned Gateway becomes unavailable, the Endpoint can be configured to find and log into another available Endpoint gateway. The TMA architecture is very flexible and makes the overall management environment more reliable. In the full Managed Node environment, the Managed Node is installed through and tightly bound to a single TMR Server. Thus, there may be less flexibility in maintaining the manageability of a Managed Node.

1.3.13 Preloaded TMA

The Tivoli Management Agent provides many benefits and is easy to maintain and control. But what about its initial installation on what could potentially include thousands of systems? Tivoli is working with a large number of hardware and software vendors to help ensure that the TMA is preloaded on systems you will purchase in the future.

Having the TMA preloaded is certainly the most powerful means of mass-Endpoint installation. In a system (for example an OS/2 system) that has an already preloaded the TMA module, the only operation required will be to activate the TMA one time. It will then search out and find Endpoint gateways, initiate the initial login process and join the TMR. All of this, of course, can be controlled through policy scripts and configuration options. We will talk about the preloaded TMA in the Chapter 2, “Tivoli Ready with Tivoli Management Agent” on page 31, as well as other installation options later in this redbook.

1.3.14 Functions Requiring Managed Nodes

As we mentioned, the TMA and its three-tiered management structure provides many new and improved functions. In many cases, it replaces the full Managed Node better than the prior Tivoli Management Environment. However, there are still framework and application functions that require the use of a full Managed Node. These cannot be configured to run on an Endpoint. For example, the TMR Server, RIM host, T/EC server, Software Distribution source host and others, still require the function provided by the full Managed Node. But remember, this does not prevent the installation of the TMA on these systems. So, you can still manage these systems through the Endpoint and use the Managed Node capability simply to support the functions requiring it. When planning how to structure your new TMR, you will need to consider these restrictions. We provide more information on this topic in Chapter 6, “TMA and Tivoli Management Applications” on page 211.

1.4 Management Topology with TMA

In Version 3.6 of Tivoli, the management topology changed from a two-tiered structure to a three-tiered structure. The three-tiered structure mainly provides the following advantages:

- Off-loading the TMR Server
- Increasing the number of systems a single TMR can manage
- Flexible configuration for Endpoints
- High availability for Endpoint operations
- Configuring the MDist repeater automatically

The three-tiered management structure is a natural concept for managing a large environment. According to this concept, the Endpoint gateway plays the role of mid-level manager. In other words, the Endpoint gateway takes responsibility for managing the Endpoints which have logged into the Endpoint gateway. Therefore, all requests from the Endpoints must be received by the Endpoint gateway and most of them will be processed by the Endpoint gateway instead of the TMR Server. Only requests which the Endpoint gateway cannot handle would be forwarded to the TMR Server where the TMR Server processes them. In the three-tiered structure, the manager system for the Endpoints is the Endpoint gateway, so that the Endpoint must send all requests to the Endpoint gateway.

In the prior TMR structure, the TMR Server had to manage all managed systems (Managed Nodes). This was a disadvantage when managing a large environment. In a three-tiered structure, a single Endpoint gateway can handle several hundreds of Endpoints and the TMR Server (Endpoint Manager) can handle up to 200 Endpoint gateways. What does this mean? This means we can now say there are few limitations to the number of managed systems in a single TMR. Interconnecting multiple TMRs can still be a very strategic solution for managing a large environment. However, with TMA, we have another powerful solution that has become available.

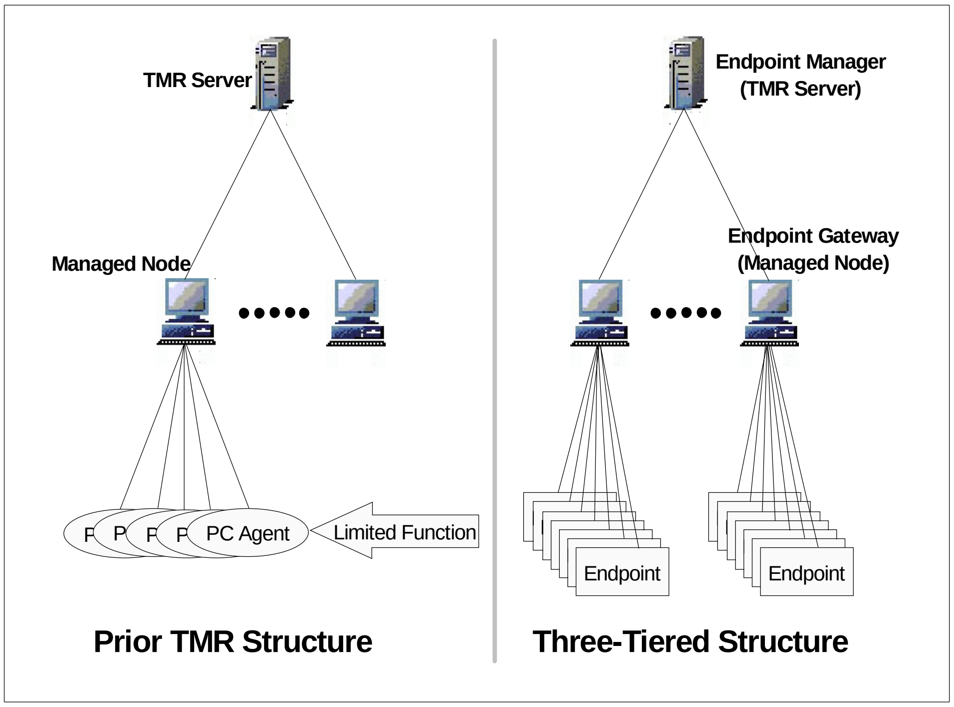


Figure 8. Three-Tiered Management Structure

If you have ever installed a Managed Node, you understand the relationship between the TMR Server and the Managed Node. Basically, the TMR Server used to install the Managed Node manages the Managed Node forever unless you reinstall or reconfigure it. In this situation, if the TMR Server becomes unavailable, all management operations will be unavailable.

Although we still have a dependency on the TMR Server in a three-tier architecture, much of the TMR Server's functions can be off-loaded to a gateway. The Endpoint has the flexibility to automatically log in through an alternate gateway if its primary gateway becomes unavailable. In this case, the available Endpoint gateway is called the alternate gateway. The Endpoint attempts to log in to the alternate gateway automatically if the Endpoint detects the assigned gateway is unavailable. This is called an isolated login. But, if you do not install or configure any alternate gateways, the Endpoint will not be able to receive management operations when the assigned gateway goes down.

Tivoli Version 3.6 provides improved functions, however, we have to understand them thoroughly if we want to use them efficiently. From this point of view, in the three-tiered environment, the following concepts are very important:

- TMR Design
- Management Resource Allocation (EP Manager, EP Gateway, Endpoint)
- Endpoint Configuration (login interfaces information)

Later in this book, we will talk about the above subjects, as well as provide hints and tips for implementation.

1.5 Summary and Future Direction

This chapter has shown the major features of the TMA. The TMA has changed many management operations in the Tivoli management environment. The TMA implementation is very powerful and will continue to be enhanced to remove any remaining limits.

For example, Tivoli plans to make the Endpoint gateway independent in the near future. This means the Endpoint gateway will no longer be required to exist on a Managed Node. This will increase your configuration options and potentially leave more resources available for Gateway operations.

As we said at the start of our discussion in this chapter, the Tivoli architecture provides stability. That is, the applications you have been using can continue to run in Version 3.6 just as they have before. In addition, you can take advantage of the benefits provided by the Tivoli Management Agent. Again, the application functions will be the same as you have become accustomed to, but they will perform better and be easier to maintain by utilizing the TMA.

Another aspect of the Tivoli architecture is its extensibility. The TMA demonstrates that the Tivoli architecture is constantly evolving to meet the demands of its customers.

The rest of this redbook primarily introduces detailed information on how to plan for, install, control, configure, and use the TMA. The information presented in this redbook is intended for Tivoli administrators and those supporting customers in the management of large distributed system environments with Tivoli and the Tivoli Management Agent.

Chapter 2. Tivoli Ready with Tivoli Management Agent

Tivoli has announced Tivoli Ready products, which provide effective enterprise management solutions and powerful competitive advantages for customers. This chapter introduces Tivoli Ready products and the preloaded Tivoli Management Agent.

2.1 Overview of Tivoli Ready

From mainframes to desktop PCs, from applications to databases, customers are building complex IT environments to support their business needs. This complexity carries significant risks. Unless the complexity is effectively managed, businesses face downtime and potential lost revenue. This is why customers invest a lot of time and money in their management systems.

Tivoli management software allows the customer to gain control and improve the performance of all the resources within their IT environment. With Tivoli products as a foundation, customers can expect to receive maximum value from their IT investments. And, return on this investment is even higher when those IT resources offer immediate, out-of-the-box manageability with Tivoli.

Tivoli works with hundreds of industry-leading hardware and software vendors to offer tools and products that can be effectively managed and extend Tivoli’s management features. This new industry, which is built around openness and cooperation, brings customers real value.

Software and hardware vendors that can provide standards-based integration with a leading management provider add significant value to their applications and devices. The Tivoli Ready logo (refer to Figure 9 on page 31) is the guarantee that products carrying this logo have passed rigorous product certification testing by Tivoli to ensure their product delivers out-of-the-box integration with Tivoli Management software. Tivoli Ready certified products take advantage of key technologies in order to extend the management services of Tivoli software to their applications and devices.

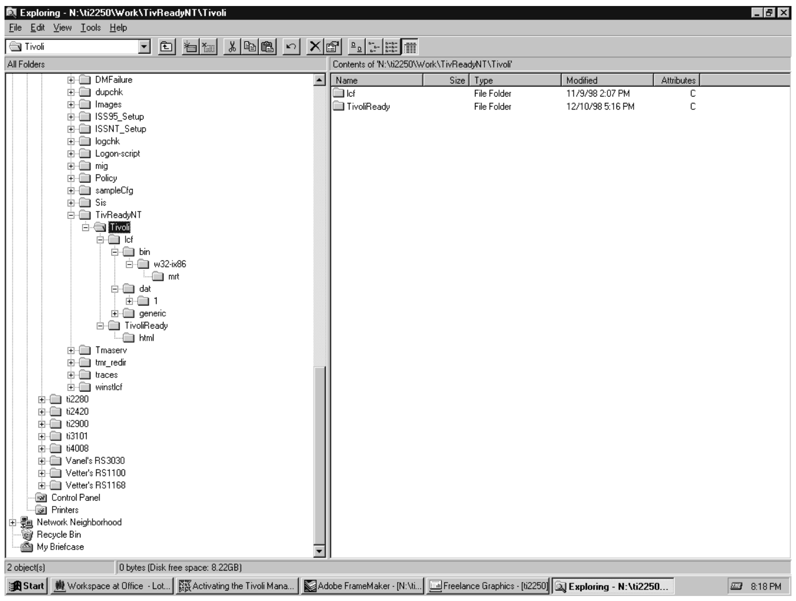
Figure 9. The Tivoli Ready Logo

Tivoli tests the Tivoli Ready products in a Tivoli-certification lab to ensure that integrated applications are dependable, work with the Tivoli Management software, and interoperate with other Tivoli Ready products. This lab duplicates a customer's distributed network environment with a variety of systems and applications.

The Tivoli Management Agent (TMA), in conjunction with the Tivoli Framework and Management Applications, provides the customer with the framework necessary to perform management operations such as software distribution, inventory, user administration and distributed monitoring. TMA is already being provided with some Tivoli Ready products. This is called preloaded TMA or Tivoli Ready with TMA. The preloaded TMA allows the customer who purchases the preloaded TMA products to immediately use the management feature provided by Tivoli. In the next section, we introduce the preloaded TMA and its advantages.

2.2 What Is Preloaded TMA?

As we mentioned above, some Tivoli Ready products already contained TMA in the their software packages. This means the `lcfd` daemon already exists on the user's hard disk. However, by default, the preloaded TMA engine (the `lcfd` daemon) does not start automatically, since it will only be of value if the Tivoli Framework is installed in the customer's environment. Therefore, when the customer has installed Tivoli Ready with TMA products on the machine, the TMA (`lcfd` daemon) exists under the C:\Tivoli\lcf (for Windows systems) directory in an inactive state. The following figure shows the directory tree of the preloaded TMA for Windows NT. As you can see, the tree structure under the Tivoli directory is the same as the normal TMA tree structure (refer to Figure 10 on page 33).



```
Exploring - N:\ti2250\Work\TivReadyNT\Tivoli

All Folders
        DMFailure
        dupchk
        Images
        ISS95_Setup
        ISSNT_Setup
        logchk
        Logon-script
        mig
        Policy
        sampleCfg
        Sis
        TivReadyNT
          Tivoli
            lcf
              bin
                w32-ix86
                  mrt
              dat
                1
              generic
            TivoliReady
              html
        Tmaserv
        tmr_redir
        traces
        winstlcf
      ti2280
      ti2420
      ti2900
      ti3101
      ti4008
      Vanel's RS3030
      Vetter's RS1100
      Vetter's RS1168
    Control Panel
    Printers
  Network Neighborhood
  Recycle Bin
  My Briefcase

Contents of 'N:\ti2250\Work\TivReadyNT\Tivoli'
Name          Size  Type         Modified            Attributes
lcf                 File Folder  11/9/98 2:07 PM     C
TivoliReady         File Folder  12/10/98 5:16 PM    C

2 object(s)    0 bytes (Disk free space: 8.22GB)
```

File Edit View Tools Help
Tivoli
Start
Workspace at Office - Lot...
Activating the Tivoli Mana...
Adobe FrameMaker - [N:\ti...
Freelance Graphics - [ti2250]
Exploring - N:\ti2250...
8:18 PM

Figure 10. The Directory Tree of Preloaded TMA for Windows NT

What is the first thing that customers who will use TMA for managing their systems should do? If the customer wishes for the system to become part of their Tivoli enterprise, the customer needs to activate or wake-up the `lcfd` daemon that has already been installed on the system. This activation process is easier than the TMA installation process. Normally, each preloaded TMA product provides the activation method, so the customer just needs to follow the activation instructions. The following figure shows how to use the preloaded TMA.

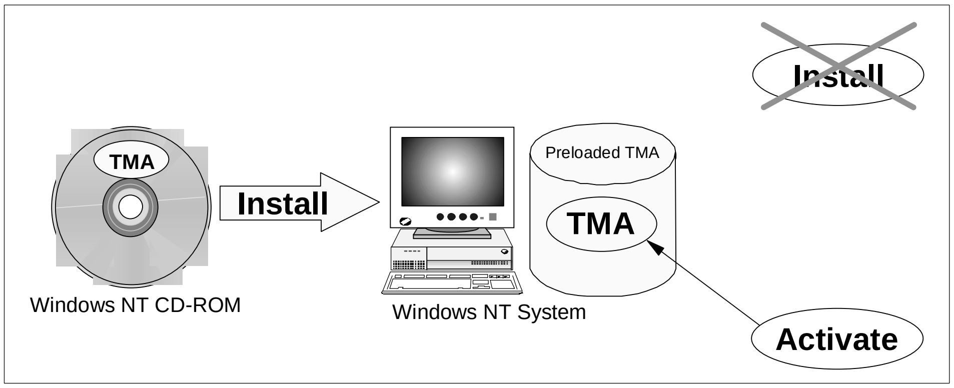


Figure 11. Activating Preloaded TMA

As you can see, the preloaded TMA does not need the Endpoint installation process and the activation process is normally a batch program that is provided by the preloaded TMA. This process only needs to be performed once. After this activation, TMA runs on the machine and is configured to start and log in to the TMR every time the machine is started.

2.3 Advantage of Preloaded TMA

The preloaded TMA makes the installation and configuration process easy and fast. First of all, the preloaded TMA does not need the installation process of the Endpoint software. Since the installation process can be a complicated process in large environments, preloaded TMA improves the implementation speed and workload dramatically. The following table shows a comparison between the preloaded TMA and normal TMA installation.

Table 2. Comparison Between Preloaded TMA and Normal TMA Installation

	Preloaded TMA	Normal TMA Installation
Order TMA Software	Not Needed	Needed
Network Traffic	None	Approximate 600 KB per Node
Endpoint Configuration	Support	Support
Mass Installation	Easy	Average
Ease of Use	High	Average

As you can see, preloaded TMA is very powerful when you want to deploy thousands of systems. By bypassing the installation process, it not only saves whatever bandwidth might be involved, but avoids any installation errors that might occur through network or human error. This means the preloaded TMA is more reliable than the normal TMA installation for mass installations and can occur independently of network conditions.

Preloaded TMA provides a sample login script which activates and configures the preloaded TMA. When enabling the TMA for large numbers of installed systems (or systems just being installed) you could use an NT login script (or equivalent) to automate the process. The following figure shows how to configure the machine as part of your Tivoli enterprise environment.

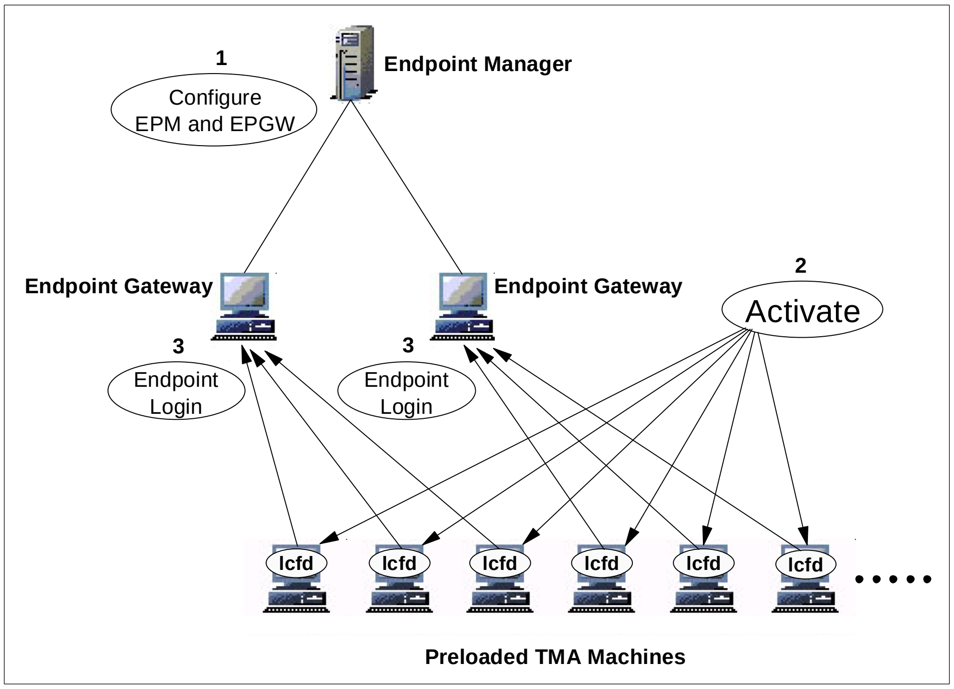


Figure 12. Implementing Preloaded TMA Machines

1. Create and configure the Endpoint Manager and Endpoint gateways.
2. Activate the preloaded TMAs. As an example, you can use an NT login script for the mass activation of Windows preloaded TMA machines.
3. The TMA logs in to the appropriate Endpoint gateway, which you configured during the activation process.

In this case, it is important that you create and configure the Endpoint Manager and Endpoint gateways before you activate the preloaded TMA activations. Then, you can activate the preloaded TMA with the appropriate options. The activation process of the preloaded TMA allows you to specify the options of the `lcfd` daemon, so that you can configure and control the Endpoint login for all of the Endpoints.

2.4 Preloaded TMA Packaging

To satisfy the customer's requirements, there are actually two different types of preloaded TMA, as follows:

• Preloaded TMA in operating systems or applications

- Zipped TMA code

The following sections introduce the characteristics of each type of preloaded TMA.

2.4.1 Preloaded TMA in Operating Systems and Applications

One type of TMA is preloaded on various operating systems or applications. We primarily talked about this type in the previous sections. As we mentioned, this type of TMA has already been preloaded on the product's CD-ROM, or on the hard disk if the operating system or application is preloaded on the hardware. Therefore, the customer who buys the operating system (OS) or application that has preloaded TMA should do the following to enable TMA:

1. Install the product (OS or application).
2. Activate the TMA.

2.4.2 Zipped TMA Code

Another type is the zipped-code package, which contains the preloaded TMA images for each platform. This zipped TMA code will be used by the customer who is planning a large-scale rollout of new machines or disks and uses the Tivoli Ready packaging to create a Golden Disk. Figure 13 on page 37 shows how the customer uses the zipped TMA code.

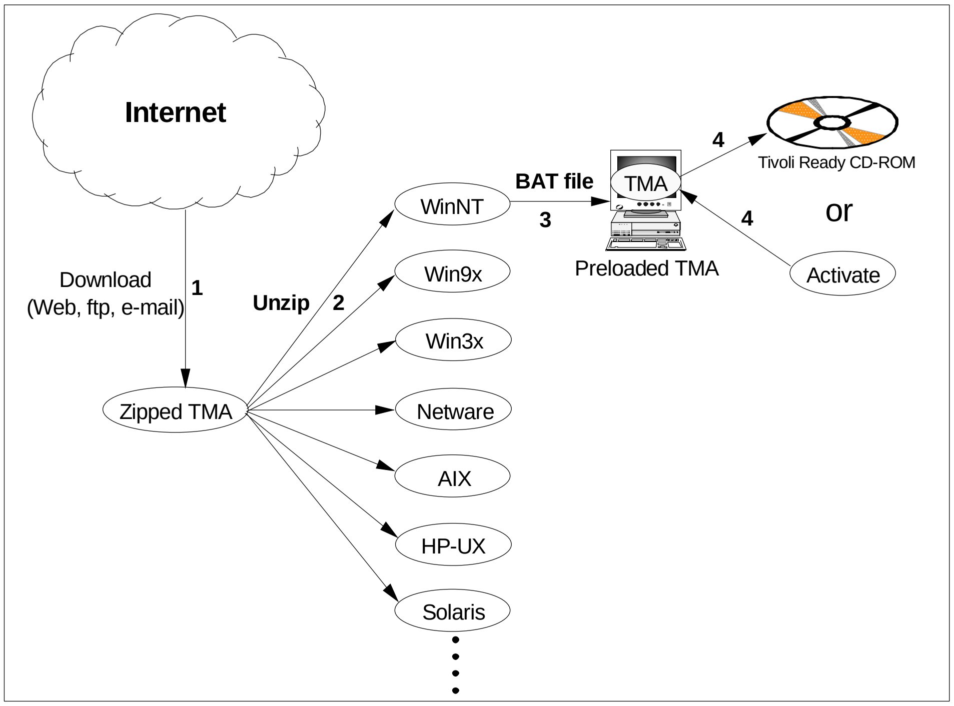
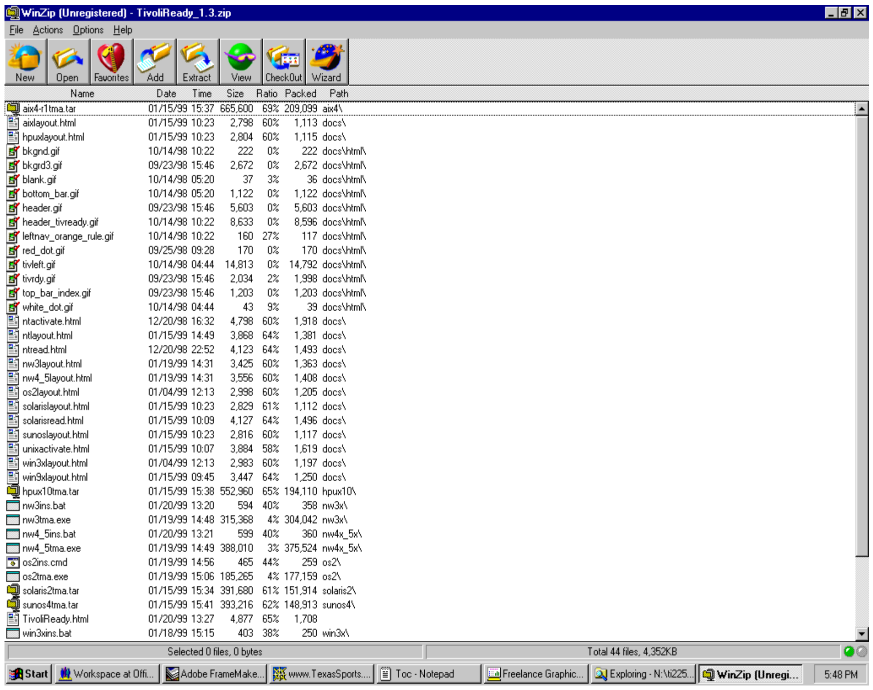


Figure 13. How to Use the Zipped TMA Code

1. The customer downloads the zipped TMA using the Web or ftp.
2. The customer unzips the zipped TMA on the machine that will be configured as the preloaded TMA prototype. After the unzip, the preloaded TMA images for each supported platform appear in the subdirectory.
3. The customer selects the appropriate preloaded TMA image and executes the batch program provided by the zipped TMA. The preloaded TMA image is unpacked, and the preloaded TMA configuration processes are completed. At this time, the `lcfd` daemon is not started because the zipped TMA creates a preloaded TMA image. Therefore, the customer needs to activate the `lcfd` daemon to use it immediately.
4. If the customer creates a Golden Disk for a large-scale deployment, this machine can be the prototype for the CD-ROM. If the customer simply wants to configure this machine as the TMA, the customer needs to complete the activation operation.

Figure 14 on page 38 shows the contents of the zipped TMA code.



Name	Date	Time	Size	Ratio	Packed	Path
aix4-r1tma.tar	01/15/99	15:37	665,600	69%	209,099	aix4\
aixlayout.html	01/15/99	10:23	2,798	60%	1,113	docs\
hpuxlayout.html	01/15/99	10:23	2,804	60%	1,115	docs\
bkgnd.gif	10/14/98	10:22	222	0%	222	docs\html\
bkgrd3.gif	09/23/98	15:46	2,672	0%	2,672	docs\html\
blank.gif	10/14/98	05:20	37	3%	36	docs\html\
bottom_bar.gif	10/14/98	05:20	1,122	0%	1,122	docs\html\
header.gif	09/23/98	15:46	5,603	0%	5,603	docs\html\
header_tivready.gif	10/14/98	10:22	8,633	0%	8,596	docs\html\
leftnav_orange_rule.gif	10/14/98	10:22	160	27%	117	docs\html\
red_dot.gif	09/25/98	09:28	170	0%	170	docs\html\
tivleft.gif	10/14/98	04:44	14,813	0%	14,792	docs\html\
tivrdy.gif	09/23/98	15:46	2,034	2%	1,998	docs\html\
top_bar_index.gif	09/23/98	15:46	1,203	0%	1,203	docs\html\
white_dot.gif	10/14/98	04:44	43	9%	39	docs\html\
ntactivate.html	12/20/98	16:32	4,798	60%	1,918	docs\
ntlayout.html	01/15/99	14:49	3,868	64%	1,381	docs\
ntread.html	12/20/98	22:52	4,123	64%	1,493	docs\
nw3layout.html	01/19/99	14:31	3,425	60%	1,363	docs\
nw4_5layout.html	01/19/99	14:31	3,556	60%	1,408	docs\
os2layout.html	01/04/99	12:13	2,998	60%	1,205	docs\
solarislayout.html	01/15/99	10:23	2,829	61%	1,112	docs\
solarisread.html	01/15/99	10:09	4,127	64%	1,496	docs\
sunoslayout.html	01/15/99	10:23	2,816	60%	1,117	docs\
unixactivate.html	01/15/99	10:07	3,884	58%	1,619	docs\
win3xlayout.html	01/04/99	12:13	2,983	60%	1,197	docs\
win9xlayout.html	01/15/99	09:45	3,447	64%	1,250	docs\
hpux10tma.tar	01/15/99	15:38	552,960	65%	194,110	hpux10\
nw3ins.bat	01/20/99	13:20	594	40%	358	nw3x\
nw3tma.exe	01/19/99	14:48	315,368	4%	304,042	nw3x\
nw4_5ins.bat	01/20/99	13:21	599	40%	360	nw4x_5x\
nw4_5tma.exe	01/19/99	14:49	388,010	3%	375,524	nw4x_5x\
os2ins.cmd	01/19/99	14:56	465	44%	259	os2\
os2tma.exe	01/19/99	15:06	185,265	4%	177,159	os2\
solaris2tma.tar	01/15/99	15:34	391,680	61%	151,914	solaris2\
sunos4tma.tar	01/15/99	15:41	393,216	62%	148,913	sunos4\
TivoliReady.html	01/20/99	13:27	4,877	65%	1,708	
win3xins.bat	01/18/99	15:15	403	38%	250	win3x\

Selected 0 files, 0 bytes

Total 44 files, 4,352KB

Figure 14. The Contents of the Zipped TMA Code

This is an overview of the zipped TMA code. As you can see, the zipped TMA code is a tool that creates a preloaded TMA machine. The current version of the zipped TMA code contains the TMA installation image for the following platforms:

- AIX V4.x
- HP-UX V10.x
- Sun Solaris V2.x
- Netware V3
- Netware V4 and V5
- Windows NT
- Windows 95 and 98
- OS/2

The zipped TMA code will be distributed from the Tivoli Web site (`http://www.tivoli.com`) and it is free. We provide more details on how to activate the preloaded TMA or how to unpack the zipped TMA code in Chapter 3, “Tivoli Management Agent Installation” on page 41.

2.5 Tivoli Ready with TMA Shipping Information

In this section, we introduce products that are currently shipping with preloaded TMA.

Table 3. Tivoli Ready with TMA Shipping Information

Maker	Products
3 Com	Fast Ethernet NICs
Novell	Netware
Intel	Network Management Adaptors, LanDesk Client Manager
IBM	IBM PC300, Intellistation, Netfinity Servers, ThinkPad, Token Ring TCI Adaptor, OS/400, OS/390, AIX, OS/2

Note

This shipping information is based on information as of the end of January 1999. This list will continuously be updated. Please check on the latest information from your Tivoli representative or the Tivoli Web site (`http://www.tivoli.com`).

2.6 Future Directions

The preloaded TMA allows systems to be integrated with the Tivoli Management Environment easily, even if there are thousands of systems in the environment. At this time, preloaded TMA is being planned for a variety of operating systems and applications. However, to manage whole-enterprise systems consistently using the Tivoli management applications, the preloaded TMA may someday support networking devices such as routers and hubs. If this becomes the case, you may see the TMA agent become integrated with hardware devices as well. Figure 15 on page 40 illustrates this future direction.

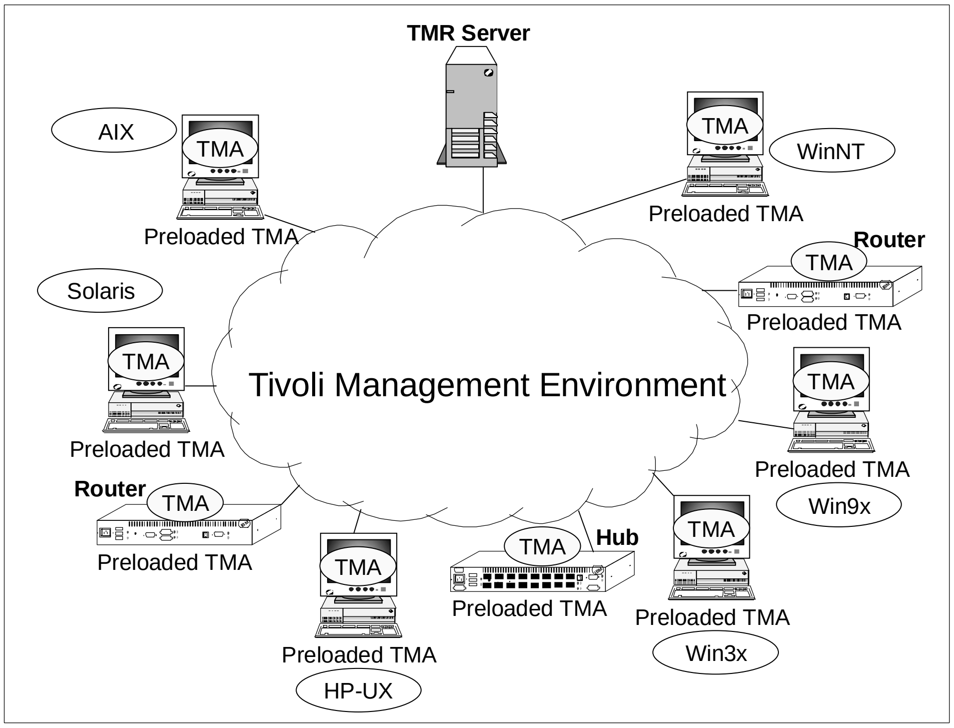


Figure 15. Integrating Network Management with Systems Management

In many environments, network management and systems management, though closely related, are being handled separately due to the available management product mix. Most customers would prefer to have an integrated solution for both network and systems management. Tivoli already provides applications that bring these disciplines close together. When TMA becomes available on networking devices, seamless management will become a true reality and customers will be able to reap significant benefits.

Chapter 3. Tivoli Management Agent Installation

In this chapter, we discuss the planning and installation of the Tivoli Management Agent. This chapter includes the following information:

- Hardware, software, and configuration requirements for Endpoints
- TMA-related considerations
- Overview of Endpoint installation and configuration
- Planning for mass installation

Please refer to the *Tivoli Framework Release Notes* and the *Tivoli Framework Planning and Installation Guide* as complete sources of information.

3.1 Planning for TMA

The following sections provide information regarding planning for the Tivoli Management Agent. This information is based on Version 3.6 of the framework released in September, 1998. Please refer to the release notes for this or later versions for the latest information.

3.1.1 Installation Prerequisites

The following is a list of the management objects required to install and use Endpoints:

- **TMR Server** - The Tivoli Management Server component includes the libraries, binaries, data files, and graphical user interfaces needed to install and manage the Tivoli environment.
- **Endpoint Manager** - The Endpoint Manager runs on the TMR Server. The Endpoint Manager maintains the information related to known Endpoints and Endpoint gateways. The Endpoint Manager is automatically installed on TMR Servers. The Endpoint Manager's primary role is to assign the Endpoint to the Endpoint gateway when the Endpoint performs the initial login. The Endpoint Manager must always be involved if an alternate Endpoint gateway will be used by an Endpoint. This might occur through explicit administrative action to migrate an Endpoint from one Gateway to another, or if an Endpoint attempts another initial login due to its primary Gateway being unavailable for any reason.
- **Endpoint Gateway** - The Endpoint gateway provides the primary interface between a set of Endpoints and the rest of the TMR. As part of this role, it also assumes some of the function previously performed by the TMR Server. By shifting a share of the management processes to the Endpoint

gateway, the TMR Servers are free to service more managed systems than with previous versions of Tivoli. A single Endpoint gateway can support communications with thousands of Endpoints. In Version 3.6 of the Tivoli Management Framework, Endpoint gateways must be installed on Managed Nodes.

- **Endpoint** - The Endpoints are managed systems taking advantage of the Lightweight Client Framework. You can gather required management information from thousands of Endpoint machines and remotely manage those machines with very little overhead. Another advantage of Endpoints is the relatively small demand they make on computer resources. The Endpoint is officially referred to as the Tivoli Management Agent (TMA) in Version 3.6 of Tivoli, however, it is still often referred to as an Endpoint in the various documentation, as well as on screen messages and the Tivoli Desktop.

3.1.2 System Requirements

This section lists the software and hardware system requirements for the Endpoint. Note that some previously supported versions of operating systems are no longer supported as Managed Nodes in Version 3.6 of the Tivoli Management Framework. Specifically, the following versions of operating systems are no longer supported by Tivoli:

- HP-UX 9.x
- AIX 3.x

3.1.2.1 TMR Server and Gateways Requirements

Depending on the number of Endpoints, some additional memory may be desired.

IBM RS/6000 and PowerPC Systems Running AIX:

- AIX 4.1.2, 4.1.3, 4.1.4, 4.1.5, 4.2, 4.2.1, 4.3, 4.3.1.
- Support of AIX is limited to AIX on IBM systems.
- The libtermcap.a in the bos.compat.termcap.
- Motif Version 1.2.
- 99.5 MB of disk space and additional swap space.
- At least 64 MB (Server) / 32 MB (Gateway) of memory.

Intel 486 or Pentium Systems Running PC Operating Systems

- Windows NT 3.51with Service Pack 5, Windows NT 4.0, or NT 4.0 with Service Pack 3

- Not supported on NT multi-user add-ons, such as WinDD or Citrix WinFrame, or on NT machines running the beta release of the Windows 95 shell or New shell
- 102.5 MB of disk space and additional swap space
- At least 48 MB (Server) / 24 MB (Gateway) of memory

HP 9000 Systems Running HP-UX

- HP 9000/700 or 800 series with PA RISC 1.1 or PA RISC 2.0
- HP-UX 10.01, 10.10, 10.20, or 11.00 with Service Pack 1
- Motif Version 1.2
- 145.5 MB of disk space and additional swap space
- At least 64 MB (Server) / 32 MB (Gateway) of memory

Sun SPARC Systems Running SunOS

- SunOS 4.1.3, 4.1.4
- OpenLook with the jumbo OpenWindows patch 100444-62 or Motif Version 1.2
- 106.5 MB of disk space and additional swap space
- At least 48 MB (Server) / 32 MB (Gateway) of memory

Sun SPARC Systems Running Solaris

- Solaris 2.4 with jumbo patch 101945-23
- Solaris 2.5, 2.5.1, 2.6
- OpenLook or Motif Version 1.2
- 94.5 MB of disk space and additional swap space
- At least 64 MB (Server) / 32 MB (Gateway) of memory

3.1.2.2 Endpoint Requirements

The Endpoint uses approximately 1 MB of memory.

IBM RS/6000 and PowerPC Systems Running AIX

- AIX 4.1.2, 4.1.3, 4.1.4, 4.1.5, 4.2, 4.2.1, 4.3, 4.3.1.
- Support of AIX is limited to AIX on IBM systems.

Intel 486 or Pentium Systems Running PC Operating Systems

- IBM PC AT-compatible machine
- Windows 3.1, or Windows for Workgroup 3.11 with the following WINSOCK compliant stacks:

    - FTP Software OnNet16 v2.5
    - Microsoft TCP/IP-32 Version 3.11
    - NetManage Chameleon 4.01, 4.5, or 4.6
    - Windows 95 or Windows 98
    - Windows NT 3.51with Service Pack 5, NT 4.0, or NT 4.0 with Service Pack 3
    - Not supported on NT multi-user add-ons, such as WinDD or Citrix WinFrame, or on NT machines running the beta release of Windows 95 shell or New shell
    - Netware 3.11, 3.12, 4.0.1, 4.1, 4.1.1
    - OS/2 3.0 and 4.0 with OS/2 TCP/IP package

HP 9000 Systems Running HP-UX

- HP 9000/700 or 800 series with PA RISC 1.1 or PA RISC 2.0
- HP-UX 10.01, 10.10, 10.20, or 11.00

Sun SPARC Systems Running SunOS

- SunOS 4.1.3, 4.1.4

Sun SPARC Systems Running Solaris

- Solaris 2.4 with jumbo patch 101945-23
- Solaris 2.5, 2.5.1, 2.6

The following list identifies the supported platforms listed in the *Tivoli Framework Planning and Installation Guide*:

- AS/400
- Digital UNIX
- DG/UX
- NCR 3000
- Pyramid MIServer-ES
- Sequent DYNIX/ptx
- SCO UnixWare
- SGI IRIX

3.1.3 TMA Planning Considerations

Some important considerations related to planning for TMA installation are described in the following section.

3.1.3.1 Communication Requirements

Tivoli's distributed architecture is designed to work across a wide variety of local area, wide area networks, and network topologies. The minimal requirement is for bidirectional, full-time, interactive TCP/IP lines.

3.1.3.2 DNS and Host Name/IP Address Mapping

The Tivoli service, or daemon, that runs on each client and on the server must be able to map a machine's IP address to a host name during the initial connection process between services. This technique is sometimes known as reverse-name mapping, and is sometimes not available in a DNS environment. Mapping between IP addresses and host names or host names and IP addresses can typically be done using one of the following sources of data:

- UNIX /etc/hosts file or NT LMHOST file
- NIS
- DNS

With the first two data sources, both forward and reverse name mapping is available. However, it is possible to configure DNS such that reverse mapping is not available. Tivoli uses both. You must have reverse mapping from the TMR Server to the client and from the client to the TMR Server.

3.1.3.3 DHCP (Dynamic Host Configuration Protocol)

The Tivoli Management Framework also supports the use of DHCP for Endpoint clients. If you have clients that require DHCP support, you must first configure DHCP in your environment. The Framework does not provide DHCP. It simply enables you to use DHCP and Tivoli in your environment.

3.1.3.4 Network Address Translation (NAT)

With the increase of Internet addresses worldwide, it is becoming more difficult for enterprises to assign globally unique IP addresses to each host. Network Address Translation (NAT) devices are fast becoming a way for large, multisite enterprises to avoid IP address conflict. Hosts within enterprises are often partitioned into public and private address spaces. The NAT device acts as a router between these address spaces with the ability to translate IP addresses and ports. In the Tivoli environment where thousands of Endpoint clients may reside in different domains of networks, the NAT device can be used to interconnect these Endpoints to the TMR Servers and the Endpoint gateways. The Tivoli NAT environment supports dynamic sharing of public IP addresses. The following are requirements for the NAT support in the Tivoli environment:

- The NAT device must act as a router for IP traffic between public and private networks.
- The NAT device must act as a proxy DNS server to resolve host name addresses in each address space.
- Fully qualified host names must be used for Endpoint gateways when providing login information.

3.1.3.5 Tivoli Management Regions

To manage thousands of resources that are geographically dispersed, the Tivoli Framework enables you to logically partition your managed resources into a series of loosely coupled Tivoli Management Regions (TMRs). There can be several reasons for creating multiple TMRs and interconnecting them, such as organizational structure, network topology and security requirements.

3.1.3.6 Endpoint Gateway Location

The primary function of the Endpoint gateway is to serve as a bridge between the TMR Server and the Endpoint. By controlling all communications with the Endpoint and all the operations that run on the Endpoint, the Endpoint gateway off-loads the work from the TMR Server.

The Endpoint gateway also serves as the MDist repeater site for profile and software distribution to the Endpoints. Locating Endpoint gateways based on your network topology has a major effect on the success of these distributions and the impact of Tivoli operations on your network.

When the information is distributed through the Endpoint gateway to Endpoints, a single data stream is sent to the Endpoint gateway first. From there, multiple streams fan out to multiple Endpoints.

Assume that you have two sites, the local site that the TMR Server resides in, and the remote site including one hundred Endpoints. Also assume they are connected by relatively slow communication links. If you put the Endpoint gateway in the local site and attempt to distribute the information to 100 systems located in the remote site, very large data streams will be pushed through the slow link 100 times.

However, if the Endpoint gateway is placed in the remote site, close to the Endpoints, and the data stream crosses the slow line only once and is pushed to 100 Endpoints using the local area network, overall performance will obviously be improved. So, putting the Endpoint gateways at the end of slow links is usually effective for transferring data.

3.1.3.7 Organizing Endpoint Gateways

In the TMA environment, each TMR Server supports up to 200 Managed Nodes, but each Endpoint gateway can support a larger number of Endpoints. The three-tiered management structure dramatically increases the number of managed systems in a single TMR. Planning the architecture of such an environment becomes one of the key points for a successful implementation.

In determining the number of Endpoints for an enterprise, it’s important to consider the quality of service desired. With the addition of the Gateways, you can support a large number of Endpoints without a performance decrease. It is important to determine the right balance between the number of Gateways and Endpoints.

Organize and distribute Endpoint gateways and Endpoints by:

- Physical location
- Type of applications running on Endpoints
- Amount of upcalls / downcalls

This will be a necessary step in planning the proper number and location of Gateways to support your Endpoints.

3.1.3.8 Endpoint Gateway Overhead

The Endpoint gateway does not have to be a dedicated system. The performance impact on the Endpoint gateway depends on the number of Endpoints using the Endpoint gateway and the type of applications running on the Endpoints. The load on the Endpoint gateway will be greatest when applications distribute data or make downcalls. For example, an inventory scan will result in a burst of data coming back to the Endpoint gateway. If you are concerned about loading the Endpoint gateway, the inventory scan can be performed at a time when the Endpoint gateway machine is less busy. The system administrator needs to consider not only the design for the TMR and the allocation of the management resources, but also how it performs. It should be noted, that in general, Tivoli recommends that Endpoint gateways be dedicated to that task.

3.1.3.9 Recommended Numbers of Endpoints for Each Gateway

Tivoli recommends no more than 10,000 Endpoints per single TMR and no more than 2,000 Endpoints per Endpoint gateway. However, keep in mind the numbers depend heavily on your network configuration and topology. For example, you may never want to have an Endpoint gateway service 2,000 Endpoints for the same reasons you would never want a Managed Node repeater servicing that many Endpoints. Therefore, the appropriate numbers of Endpoints are different for each customer.

3.1.3.10 The Limit of 200 Gateways per TMR

In Version 3.6 of Tivoli Management Framework, the Endpoint gateways must run on the Managed Node. Due to several factors, including the overhead of maintaining a distributed database across many nodes, the number of Managed Nodes supported in a single TMR is about 200. You are still bound to this limit, and therefore, there is the same limit on the number of Gateways that can be supported in a single TMR.

3.1.3.11 Firewall Support

Version 3.6 of the Tivoli Endpoint client will not work correctly through firewalls. A firewall is not supported between the Endpoint gateway and the Endpoint.

3.2 TMA Installation

The following sections describe the installation of Endpoints and their prerequisites.

3.2.1 Overview of TMA Installation

To utilize the Tivoli Management Agent, you must have both an Endpoint Manager and an Endpoint gateway installed and configured properly. In Version 3.6 of the Tivoli Management Framework, the Endpoint Manager must run on the TMR Server. The Endpoint gateway must run on a Managed Node.

3.2.1.1 Endpoint Manager Installation

As we mentioned, the Endpoint Manager is automatically installed when the TMR Server is installed. We will not review the details of installing the TMR Server. This is well documented in the *Tivoli Framework Planning and Installation Guide*. Once the installation is complete, the Endpoint Manager will be available. If you are familiar with Version 3.1 of the Tivoli Management Framework, you will notice only one significant difference on the Tivoli Desktop in a 3.6 installation: the Endpoint Manager icon.

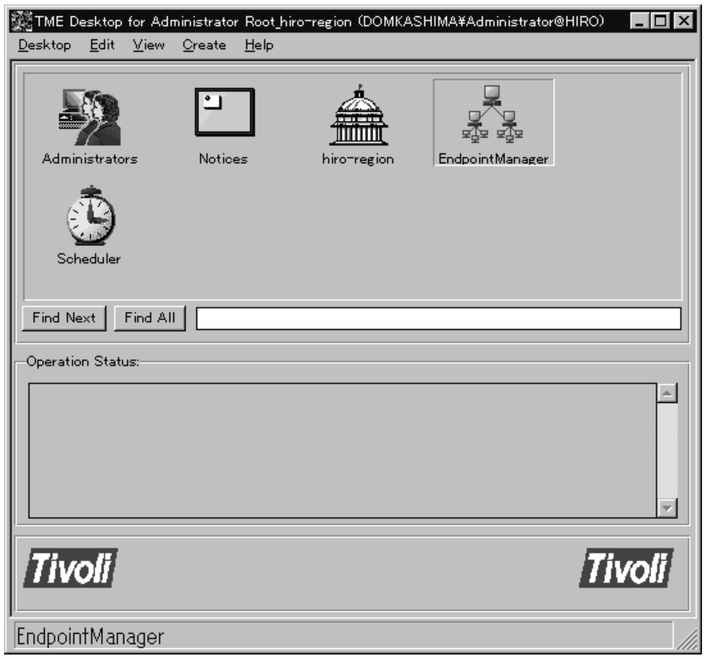


Figure 16. Endpoint Manager Icon on the Desktop

From the command line interface, you can verify the Endpoint Manager exists by issuing the `wls` command. An example of this command follows:

```
C:\Tivoli>wls
Notices
Administrators
hiro-region
EndpointManager
Scheduler
C:\Tivoli>
```

3.2.1.2 Managed Node Installation

The Endpoint gateways must be created on a Managed Node or on the TMR Server. Except for very small or test environments, it is typically not recommneded to install a Gateway on the TMR Server, as this goes against one of the goals of the architecture which is to off-load the TMR Server. This means, of course, we have to create the Managed Node, which will be configured as the Endpoint gateway before the Endpoint gateway installation. Several ways to install a Managed Node are described in the following sections:

Local Installation (NT Managed Node Only)

You can setup the Windows NT Managed Node locally from the Tivoli Management Framework CD-ROM. After this operation, you can install the Endpoint gateway using the Desktop interface or the `wcrtgate` command. Both operations are able to be performed on the target Managed Node (the Desktop module should be installed on the target node when you use the Desktop interface). The commands to install the Managed Node and the Desktop are:

NT Managed Node	`setup client`
Tivoli Desktop	`pc/desktop/disk1/setup`

Installation from TMR Server

You can install the Managed Nodes remotely from TMR Server. Since you can install all the Managed Nodes from a single machine, this is one of the most convenient methods of installing Managed Nodes. The install_client authorization role for the administrator is required for this operation.

On the Tivoli Desktop, double-click on the policy region icon in which you want to create the Managed Node. On the policy region window, select the **Create** menu to display the Managed Node Installation dialog. You can create the Managed Node from this dialog. Before you install the framework on Windows NT systems, you must install the Tivoli Remote Execution Service (TRIP) on a single NT system in your TMR. If your TMR Server is Windows NT, this step is performed automatically. Please refer to the *Tivoli Framework Planning and Installation Guide* for more information.

The `wclient` command can also be used to install Managed Nodes remotely from the TMR Server. Please refer to the *Tivoli Framework Reference Manual* for more information.

3.2.1.3 Endpoint Gateway Installation

To install an Endpoint gateway onto an already existing Managed Node, you can use the following methods. The senior authorization role is required for these operations.

SIS

Tivoli Software Installation Service (SIS) enables you to install multiple Endpoint gateways onto multiple systems in parallel. SIS performs a prerequisite check and user specified prerequisite check if it is defined. SIS also enables you to install TRIP, a Managed Node, and Tivoli Management Applications. Please refer to the *Tivoli Software Installation Service User's Guide* for more information.

Tivoli Desktop
From the Tivoli Desktop interface, right-click the **Endpoint Manager** icon and select the **Create Gateway** menu. A pop-up panel appears and you can install the Endpoint gateway to the specified Managed Node that has already been created (refer to the Figure 17).

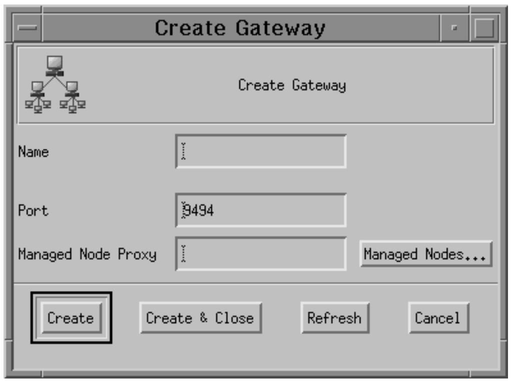


Figure 17. Create Gateway Panel

wcrtgate Command
The `wcrtgate` command creates the new Endpoint gateway on the specified Managed Node. If you do not specify the name of the Managed Node, the Endpoint gateway is created on the Managed Node from which the command was invoked. Please refer to the *Tivoli Framework Reference Manual* for more information.

3.2.1.4 Endpoint Installation

There are also several ways to install the Endpoints. We introduce them in this section.

SIS
Tivoli Software Installation Service (SIS) enables you to install Endpoints onto multiple systems in parallel. SIS performs a prerequisite check and user specified prerequisite check if defined. Please refer to the *Tivoli Software Installation Service User's Guide* for more information.

winstlcf Command
The `winstlcf` command installs and launches the Endpoint on one or more systems. This command can install UNIX and Windows NT Endpoints only. Since Windows NT does not have a remote execution service, the first Windows NT Endpoint in the domain must be manually installed using the InstallShield. You can then use the Endpoint as the proxy to install all other

Windows NT Endpoints within the same domain. Please refer to the *Tivoli Framework Reference Manual* for more information.

Local Install (InstallShield)
You can locally install and launch the Endpoint on Windows and OS/2 machines. The InstallShield provides the silent installation capability to automate the installation. Please refer to Chapter *3.5.1, "Using Silent Installation" on page 83*.

NT Login Script
A login script can be used to install and launch the Endpoint when the Windows or OS/2 user logs into an NT domain. You need to define the login script configuration on the Windows NT domain controller machine before the Endpoint installation.

TMASERV
The Tivoli Migration Toolkit provides the TMASERV (Tivoli Management Agent Login Service) to simplify the Endpoint installation. The TMASERV is provided as one of the NT services called tmaserv. It enables us to automate the Endpoint installation onto systems which log into the Windows NT domain. The TMASERV is similar to the NT login script, but more sophisticated.

Tivoli Ready with TMA (Preloaded TMA)
As we mentioned in "Tivoli Ready with Tivoli Management Agent" on page 31, the TMA module has been already installed in some products. These are called preloaded TMA. The preloaded TMA does not require the installation process. It requires only the activation process to launch the Endpoint. You can perform this activation process manually or using a login script.

3.2.2 TMA Installation Comparison

We compare these installation methods to clarify the difference. When you install Endpoints to a large number of systems, the installation method you choose becomes very important.

Note

The information we introduce in this section should help you decide the installation method most appropriate for you, but the Endpoint installation method you use may depend on a large number of factors.

3.2.2.1 Supported Platforms for TMA

Supported platform types for the Endpoint are as follows:

Table 4. Endpoint Support

	SIS	winstlcf	Login script	TMA SERV	Local	Preload
UNIX	O	O				O
Windows NT	O	O	O	O	O	O
Windows 95/98	O		O	O	O	N/A
Windows 3.x	O		O	O	O	N/A
OS/2			O		O	O

3.2.2.2 Installation Prerequisites

SIS requires at least one Windows NT Managed Node which contains the TRIP module in the TMR to install the Endpoint software to the target Windows NT machine that does not contain the TRIP module.

The `winstlcf` command requires at least one Windows NT Endpoint in the TMR to install to a target Windows NT machine.

Normally, the TMASERV requires one Windows NT TMR Server and one TMASERV server, which also has the NT Resource Kit installed. However, you can use the AIX TMR Server instead of the Windows NT TMR Server if you perform certain customizations. Please refer to 3.5.2, “Using TMASERV” on page 89 for more information about the TMASERV.

The preloaded TMA does not officially require any NT Domain Controller, but if you install many preloaded TMA machines, you may want to use a facility such as the NT login script feature to activate the preloaded TMAs automatically.

Table 5. Prerequisites

	SIS	winstlcf	Login script	TMA SERV	Local	Preload
SIS Server	O					
One TRIP Machine (NT)	O					
One NT Endpoint		O				
NT Domain Controller			O	O		O *1
Windows NT TMR Server				O *2		

	SIS	winstlcf	Login script	TMA SERV	Local	Preload
Windows NT TMASERVer				O *3		

1. The NT login script enables you to activate the preloaded TMA automatically, but it is not mandatory.
2. The AIX TMR Server is also available if you customize.
3. The Windows NT Resource Kit should be installed on the TMASERV server as a prerequisite.

3.2.2.3 Network Considerations

All of these installation methods utilize a small amout of network bandwidth to activate the Endpoint because the Endpoint daemon (`lcfd`) attempts to log into the Endpoint gateway at the Endpoint installation.

The local installation and preloaded TMA do not require any network traffic to transfer the installation image. This might be important when installing (or activating) a large number of machines.

SIS and the `winstlcf` command are examples of a push operation where explicit actions on the part of an administrator push the Endpoint code to one or more systems. When pushing to multiple systems, especially using SIS, the installation and subsequent initialization and logins may occur in parallel.

The other installation methods are based on the NT login script feature. The installation image is pulled from a file server, such as an NT domain controller, when the user logs into the domain and each Endpoint installation would be performed asynchronously. These installation methods are referred to as pull operations.

The SIS and the `winstlcf` command can take advantage of the MDist fan out function to reduce the network traffic. In other installation methods, a replicated file server may be used to reduce network traffic on slow links.

Table 6. Network Related

	SIS	winstlcf	Login script	TMA SERV	Local	Preload
No Traffic to Transfer Image					O	O
Push from TMR Server	O	O				
Pull from Endpoint			O	O		
MDist Fan Out Function	O	O				

	SIS	winstlcf	Login script	TMA SERV	Local	Preload
Can be Replicated			O	O		
Endpoint should be Started	O	O				

3.2.2.4 Installation Options

All of these installation methods provide the capability to pass arguments to the Endpoint daemon (`lcfd`), so that you can specify any installation options you require for Endpoint installation. SIS also provides enhanced prerequisite checking to help ensure a successful installation.

Table 7. Installation Options

	SIS	winstlcf	Login script	TMA SERV	Local	Preload
Gateway Address	O	O	O	O	O	O
Disable Broadcast	O	O	O	O	O	O
Installation Location	O	O	O	O	O	O
Prerequisite Check	O					

3.2.2.5 Ease of Use

Installing an Endpoint using the Endpoint installation methods we are talking about, typically requires some level of customization. For example, the NT login script will normally need to be modified for a specific environment.

The TMASERV option provides more function than the simpl NT login script, but is also somewhat more complex to install and configure.

In contrast, when using SIS, the various installation options can be specified through the SIS GUI, which may be simpler for a less experienced administrator.

The `winstlcf` command might appear to be the simplest option for installing just a few Endpoints. However, it has many parameters and may provide less information if the installation fails. Read the latest *Tivoli Framework Release Notes* before using the `winstlcf` command.

Table 8. Ease of Use

	SIS	winstlcf	Login script	TMA SERV	Local	Preload
Server Setup	A	B	C	D	N/A	N/A

	SIS	winstlcf	Login script	TMA SERV	Local	Preload
Client Installation	B	C	A	A	B	A
Failure Management	C	D	D	C	N/A	N/A
Reliability	C	D	C	C	C	C

A	Very Good
B	Good
C	Average
D	Poor

3.2.2.6 When Is It Used?

In this section, we summarize the Endpoint installation methods we have been discussing:

Table 9. When to Use Endpoint Installation Methods

	SIS	winstlcf	Login script	TMA SERV	Local	Preload
Small Number of Clients	O	O			O	O
Many Clients	O	O	O	O		O
Mass	O		O	O		O
Need Prerequisite Check	O					O
Use for UNIX and Windows	O					O
Central Management	O					

SIS provides an extensive prerequisite checking feature that checks the installation prerequisites before installing each product. In addition, SIS enables you to perform initial prerequisite checking without actually installing any products. This makes the installation reliable and, as a result, saves installation time. You can also configure additional prerequisite checks that are specific to your environment. SIS also has a response file option that enables you to install Tivoli products from the command line. This feature is particularly useful when you install a product onto a large number of machines. SIS creates a installation repository (IR) that stores the Tivoli products, patches images, and imports the installation images before they are installed. The installation images imported into the IR can be used many times.

Therefore, if you install both UNIX Endpoints and Windows Endpoints, we recommend that you think about SIS first, since it supports not only Windows but also the UNIX platforms (as compared to an NT login script, for instance). You can install many Endpoints and other Tivoli products at the same time and monitor the status of these installations from the SIS console.

However, if you install large numbers of Endpoints onto only Windows machines, you may want to to use an NT login script. It is the most powerful installation method in the Windows environment and automates the installation on new nodes entering the network without requiring administator action.

If you install just a few Endpoints, the `winstlcf` command or local installation should be quick and easy. Note that these installation methods do not provide prerequisite checking or confirmation that the Endpoint has actually started and logged into the TMR. A local installation method, such as InstallShield, depends on the individual at the workstation entering the proper paramters for their environment.

The preloaded TMA is, of course, the most powerful solution. The Endpoint module has already been installed so we simply need to activate it. However, this depends on your environment and situation. If you preload other software on systems deployed in your environment, you should certainly consider the preloaded TMA.

Note

When using SIS to install Endpoints to currently existing Managed Nodes, SIS automatically sets the alias attribute of the Endpoint to the object ID (OID) of the Managed Node. The Migration Toolkit uses this alias to help migrate subscription lists. To set the Endpoint alias manually, you can use the `wsetepalias` command provided by the Tivoli Migration Toolkit, or the `wep set alias` option. When installing in such an environment, SIS offers a great advantage in preparing for the migration.

3.2.3 Overview of Installation Options

In this section, we introduce the considerations for Endpoint installation options and other miscellaneous considerations before and during the Endpoint installation process.

3.2.3.1 Installation User

Installing the Endpoint does not require the Tivoli administrator authorization role required for the installation of other framework components. When you perform the Endpoint installation, you should log in to the operating system as:

- A member of the administrator group on Windows NT
- The root user on UNIX

3.2.3.2 Installation Locations

The following table shows the default installation directories on the Endpoint for each supported operating system.

Table 10. Installation Locations

Target	Installation Method	Default Location
UNIX	Local/winstlcf	`/opt/Tivoli/lcf`
Windows	SIS	`C:\Tivoli\lcf`
	winstlcf	`C:\Program Files\Tivoli\lcf`
	InstallShield	`C:\Program Files\Tivoli\lcf`
	Login Script	`C:\Program Files\Tivoli\lcf`
	TMASERV	`C:\Program Files\Tivoli\lcf`
	Preloaded TMA	`%SystemDrive%\Tivoli\lcf`

We strongly recommend that you specify the C:\Tivoli\lcf directory as the Endpoint installation directory instead of C:\Program Files\Tivoli\lcf on the Windows platform. If you use the directory name including a blank character, such as C:\Program Files\Tivoli\lcf, some tools may not work correctly. By default, the Endpoint installation directory contains the blank character.

3.2.3.3 Endpoint Gateway Address

In Version 3.6 of the Tivoli Management Framework, if there are multiple Gateways on the same subnet, and the Endpoint is allowed to perform the broadcast for the initial login, several of the Endpoint gateways may receive the login request and process it. It is better for you to specify the login interfaces list using the -g option as the installation option. This directs the Endpoint to contact a specific Gateway instead of broadcasting to find one.

3.2.3.4 Version Dependencies

If you are upgrading your TMR, the TMR Server, Managed Nodes, Endpoint gateways and Endpoints should be upgraded to Version 3.6 before any other

Tivoli applications are upgraded. The Endpoint Manager (TMR Server) and the Endpoint gateway must be at the same version. The Endpoints can be a lower version, but the upgrading would be performed automatically at the first connection to the Endpoint gateway if you configure the auto upgrade function.

3.2.3.5 Installation Steps

There are two steps to Endpoint installation. One is the installation of the Endpoint software to the target machine and another is the activation of the Endpoint daemon process called `lcfd` (in the Windows environment, the `lcfd` process is also activated to display a small icon on your taskbar's system tray). The initial login process is performed after the activation. The Endpoint login process can be a complicated, and we will describe it in detail in "Configuring the TMA Environment" on page 105 and "Anatomy of TMA Behavior" on page 135.

3.2.4 How to Specify Installation Option

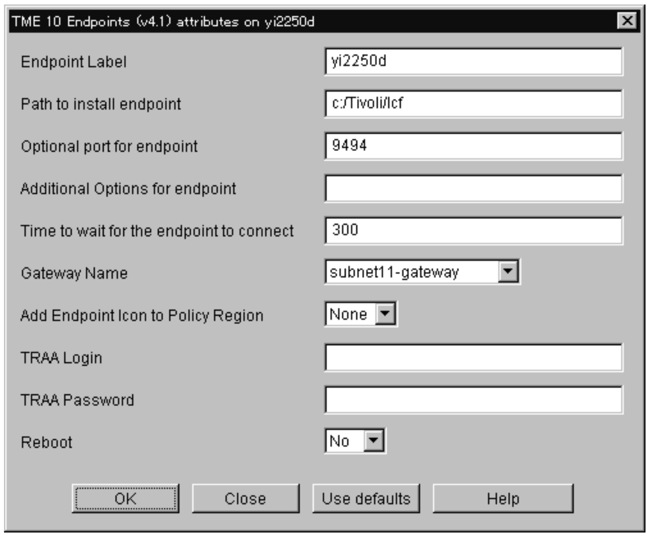
The Endpoint (`lcfd`) options which are specified at Endpoint installation are very important. In a large environment, they are very important because once you configure the Endpoint options for a mass installation, it is very difficult to modify the options. From this point of view, you should understand how to specify the Endpoint installation options. In this section, we introduce an overview of how to specify the installation options for each Endpoint installation method. For more detailed information, refer to the appropriate Tivoli manuals.

Important

The most important Endpoint option is the login interface information. When you install masses of Endpoints in a large environment, you have to consider the Gateway assignment for each Endpoint. The proper Gateway assignment makes the installation and initial login more reliable.

3.2.4.1 SIS

SIS provides the following panel to specify the Endpoint installation options. The Additional Options for Endpoint field passes the configuration arguments to the `lcfd` daemon at the activation of the Endpoint.


TME 10 Endpoints (v4.1) attributes on yi2250d

Endpoint Label	yi2250d
Path to install endpoint	c:/Tivoli/lcf
Optional port for endpoint	9494
Additional Options for endpoint	
Time to wait for the endpoint to connect	300
Gateway Name	subnet11-gateway
Add Endpoint Icon to Policy Region	None
TRAA Login	
TRAA Password	
Reboot	No

Figure 18. SIS Dialog for Endpoint Attributes

Please refer to the *Tivoli Software Installation Service User's Guide* for more information.

3.2.4.2 winstlcf Command

The `winstlcf` command provides command-line options to specify the `lcfd` daemon's parameters as follows:

```
winstlcf -d C:/Tivoli/lcf -g kodiak+9494 -L "-d3" salmon
```

where

`-d`	Specifies the target directory where the Endpoint is installed
`-g`	Specifies the IP address or hostname and port number of the Endpoint gateways
`-L`	Passes the configuration arguments to the `lcfd` command for starting the Endpoint

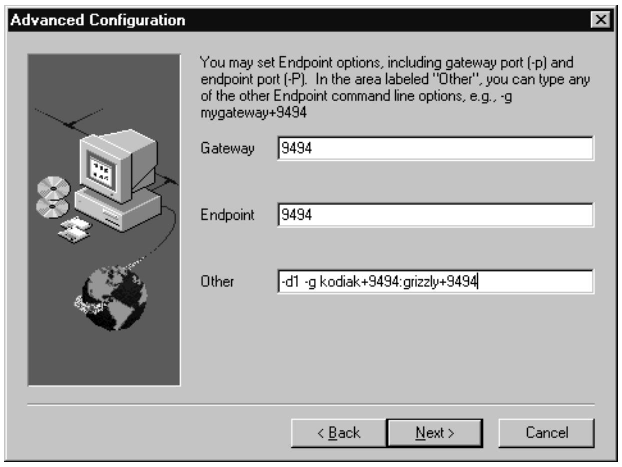
Other options are also available. Please refer to the *Tivoli Framework Reference Manual* for more information.

3.2.4.3 Local Install (InstallShield)

There are a few ways to specify the installation options when using the InstallShield.

Advanced option display

To specify the options interactively, `setup.exe` provides the following panel at the Endpoint installation.



You may set Endpoint options, including gateway port (-p) and endpoint port (-P). In the area labeled "Other", you can type any of the other Endpoint command line options, e.g., -g mygateway+9494

Gateway: 9494

Endpoint: 9494

Other: -d1 -g kodiak+9494:grizzly+9494

Figure 19. Endpoint Options in InstallShield

setup.iss File for Silent Installation

The InstallShield provides silent installation capability to automate the installation. You can use silent installation by specifying the -s option (`setup -s`). If you use silent installation, you need to modify the default setup.iss file to specify the installation options that you would like to use in your environment. The following shows a sample of the setup.iss file:

```
[SdComponentDialog-0]
szDir=C:\Tivoli\lcf
Component-type=string
Component-count=1
Component-0=TME 10 Endpoint
Result=1

[SdShowDlgEdit3-0]
szEdit1=9494
szEdit2=9494
szEdit3=-d 1 -g kodiak+9494:grizzly+9494
Result=1
```

The `szDir` specifies the target directory where the Endpoint is installed. The `szEdit3` passes the Endpoint configuration arguments to the `lcfd` command for starting the Endpoint.

You can also modify the setup.iss file through the GUI provided by the `setup.exe`. For more information to modify the setup.iss file with the GUI, please refer to “Using Silent Installation” on page 83.

3.2.4.4 Login Script

The login script uses the silent installation capability of InstallShield. If you would like to specify installation options, follow the instructions for the silent installation we mentioned.

3.2.4.5 TMASERV

The TMASERV uses the lcf_seed file to specify the installation options. The following is a sample of the lcf_seed file.

```
timeout=300
base_dir=c:\Tivoli\lcf
lcf_opts=-d1 -g kodiak+9494:grizzly+9494
```

The `base_dir` specifies the target directory where the Endpoint is installed. The `lcf_opts` passes configuration arguments to the `lcfd` daemon for starting the Endpoint. Please refer to Chapter 3.5.2, “Using TMASERV” on page 89 for more information.

3.2.4.6 Preloaded TMA

Normally, we use the login script feature to activate the preloaded TMA. If you would like to specify installation options, modify the script defined in the NT login script. Please refer to Chapter 3.4, “Configuring Preloaded TMA” on page 73 for more information.

3.2.5 Removing Endpoint Software

Removing or uninstalling the Endpoint software depends on which operating system the Endpoint is installed. However, removing the Endpoint software does not mean deleting the Endpoint entry from the Tivoli object database. To delete the Endpoint completely, you also have to remove the Endpoint entry from the Tivoli database using the `wdelep` command. Again, removing the Endpoint software is different from deleting the Endpoint from the Tivoli database.

How to remove the Endpoint software is different for each platform where the Endpoint is running, so you need to perform specific steps for each case. The following introduces how to remove the AIX, Windows NT, and Windows 95/98 Endpoints. For removing the Endpoint from other platform types, please refer to the *Tivoli Planning and Installation Guide*.

3.2.5.1 AIX

To remove the Endpoint from the AIX:

1. Stop the Endpoint daemon with following command:

```
/opt/Tivoli/lcf/dat/1/lcfd.sh stop
```

2. Remove the Endpoint installation directory and subdirectories. The default location is /opt/Tivoli/lcf.
3. Remove the Endpoint environment directory, and embedded files and subdirectory. The directory is /etc/Tivoli/lcf. The environment commands are located in lcf_env.sh and lcf_env.csh.
4. Remove the Endpoint startup entry from /etc/inittab using the following command:

```
/etc/rmitab rctma1
```

5. Remove the /etc/rc.tma1 and /etc/inittab.before.tma1 files.
6. Remove the /etc/Tivoli/*/userlink.htm file.

3.2.5.2 Windows NT/95/98 (Using the uninst.bat Command)

If you can find `C:\Tivoli\lcf\uninst.bat` command, follow these directions:

1. Stop and remove the Endpoint service by issuing the following command:

```
C:\Tivoli\lcf\uninst
```

2. Remove the Endpoint environment directory, subdirectory, and files. The directory and files are:

   C:\Winnt\Tivoli\lcf, lcf_env.sh and lcf_env.cmd (for Windows NT)

   C:\Windows\Tivoli\lcf and lcf_env.bat (for Windows 95/98)

3. Remove the c:\Etc\Tivoli\userlink.htm file.

Windows NT/95/98 (Without the uninst.bat Command)

The `wlcfinst`, the TMASERV and the preloaded TMA do not provide the `uninst.bat` utility.

1. Stop and remove the Endpoint service by issuing the following command:

```
C:\Tivoli\lcf\bin\w32-ix86\mrt\lcfd -r "lcfd"
```

2. Execute the following command to terminate the `lcfep` process:

```
C:\Tivoli\lcf\bin\w32-ix86\mrt\lcfep -s
```

3. Remove the Endpoint installation directory and subdirectories. The default location is C:\Tivoli\lcf or C:\Program Files\Tivoli\lcf.

4. Remove the Endpoint environment directory, subdirectory, and files. The directory and files are:

   C:\Winnt\Tivoli\lcf, lcf_env.sh and lcf_env.cmd (for Windows NT)

   C:\Windows\Tivoli\lcf and lcf_env.bat (for Windows 95/98)
5. Remove the C:\Etc\Tivoli\userlink.htm file.
6. Edit the Windows 95/98 registry (Windows 95/98 only):
   - Start the `regedit` program using the run program feature.
   - Expand the registry to the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices key.
   - Delete the `lcfd` entry.

3.3 Planning for Mass Installation

Installing the Endpoint on a handfull of systems can easily be accomplished on an individual system basis. However, if you are working with many Endpoints, meaning more than one hundred, we consider a mass installation. You should use the Endpoint installation method we introduced, or else install the preloaded TMA. There are some considerations to remember for mass installation. In this section, we introduce considerations for performing a mass installation.

3.3.1 Using NT Login Script

As we mentioned, if you install thousands of Endpoints to Windows machines, using the NT login script is one of the most powerful ways. In this section, we introduce an example of a mass installation using the login script.

3.3.1.1 Creating Installation User in NT Domain

The NT login script can be configured for each user in the domain, but it may be complicated for you to configure the script which performs the Endpoint installation and specify the Endpoint options for each user in the domain. One consideration is that for Windows NT, the user under which the installation process runs must be part of the NT administrator’s group. In this situation, creating a special installation user is one solution. In this case, create a user to install the Endpoint, and each machine shares the same user temporarily. When you create the installation user, we recommend that you configure the user attributes in the User Properties panel of the Windows NT User Manager for Domains utility as follows:

User Must Change Password at Next Logon	Disable

User Cannot Change Password	Enable

The most important thing here is that each person who uses the machine where the Endpoint will be installed has to know the installation user and has to log into the domain as the installation user at least once. Of course, after the user logs into the domain as the installation user, they can change the login user from the installation user to the original user by performing a logout and login. The following figure shows this process.

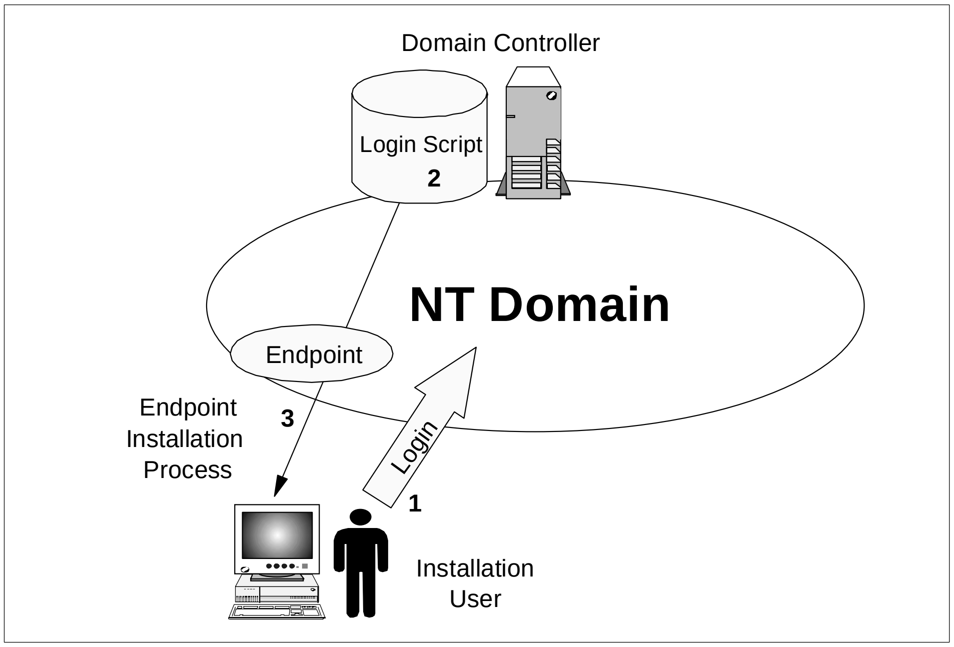


Figure 20. The Mass Installation with the NT Login Script

1. The user of the machine where the Endpoint will be installed attempts to log in to the domain as the Endpoint installation user.
2. The domain controller executes the login script defined in the user profile of the installation user.
3. The login script performs the Endpoint installation process. As a result, the Endpoint software is installed to the machine and it becomes an Endpoint. Then it attempts to perform the initial login to the appropriate Endpoint gateway.

3.3.1.2 Sample Login Script

We developed a sample login script for Endpoint mass installation during the creation of this redbook. This script and its use is described below.

The Endpoint can be installed onto a Windows machine using the login script when the user logs in to the domain. The login script checks whether the Endpoint software has already been installed or not and, if necessary, launches the Endpoint installation process. This script supports Windows 95/98 and Windows NT machines and has the option to report on the completion of the Endpoint login process to the user. Without this option, the user cannot know that the Endpoint installation finished successfully or terminated abnormally on the Endpoint machine. If you enable this option, the user can check the status of the installation process and the log file (lcfd.log) if the installation fails.

We will provide detailed information about how to enable the option that reports on the completion of the Endpoint login process in Chapter 3.3.2.2, “Completion Checking from Endpoint” on page 69.

```
REM @echo off
REM
REM Sample LCF logon script.  Call this script from the user’s normal logon script.
REM
REM Using logon scripts:
REM 1) Change the SERVER and INSTALLDIR settings to point to the lcf-images directory
REM  on the network.
REM 2) On the NT Domain server, copy this script to the scripts directory. By default
REM this directory is ’C:\WINNT\system32\Repl\Import\Scripts’.  This script can
REM    also be replicated automatically from primary domain controllers, but you can
REM    read about that fun stuff yourself.
REM 3) Run the User Manager program, select a user and then select "profile". Enter
REM    the name of this script in the "Logon scripts" area.
REM 4) Whenever that user logs in to this domain, this installation script will run
REM on their machine.
REM

REM Script settings - Modify these to your environment
set MAPDRIVE=Y
set SERVER=\\grizzly\exports
set INSTALLDIR=\Tivoli36\lcf
set DESTDIR=c:\Tivoli\lcf
set DESTDIR2=c:\"Program Files"\Tivoli\lcf
set LOGDEST=nul
set ISHIELDOPTS=-s

REM Optional introduction for user
echo Installing Tivoli Lightweight Client Framework > %LOGDEST%

REM Figure out which interp
REM @@changed 98/11/09 ver | find "Windows 95" > nul
ver | find "Windows 9" > nul
if errorlevel 1 goto checkNT
set INTERP=win95
goto install

:checkNT
```

```
ver | find "Windows NT" > nul
if errorlevel 1 goto assume3x
REM @@changed 98/01/05 INTERP=w32-ix86
set INTERP=winnt
goto install

:assume3x
set INTERP=win3x

:install
REM Check for previous installation
if exist %DESTDIR%\bin goto prevInst
if exist %DESTDIR2%\bin goto prevInst

:doit
echo OS Detected as: %INTERP%. > %LOGDEST%

REM Map drive to get to installshield files
net use %MAPDRIVE%: /DELETE > nul
net use %MAPDRIVE%: %SERVER%

REM Execute installshield setup
start %MAPDRIVE%:%INSTALLDIR%\%INTERP%\setup.exe %ISHIELDOPTS%
echo Successfully installed for %USERNAME% > %LOGDEST%

REM Start log checker
REM start sh logchk
goto end

:prevInst
echo Script detects that LCF is already installed > %LOGDEST%

:end
pause
```

Note

When you install the Endpoint onto the Windows machine, a small icon appears on the task bar. The `lcfep.exe` process displays this icon. However, it is a separate process which is installed and executed regardless of the success of the actual Endpoint installation and login. Therefore, do not use the appearance of this icon as an indication of a successful installation.

3.3.2 Installation Completion Check

In this section we investigate methods for determining the successful installation and login of Endpoints. Developing such a method is critical to the successful automated installation of a large number of Endpoints.

3.3.2.1 Completion Checking by Administrator

In an Endpoint mass installation, how do you know the Endpoint installation has completed successfully? How do you attempt to install thousands of Endpoints? The most efficient and simple way to check the status of an

Endpoint is the wep command. The wep command enables us to check the existence of the Endpoint and Endpoint gateway. The following shows sample output of the wep command:

```
/>wep ls
G      1189622596.4.21  grizzly-gateway
G      1189622596.2.19  kodiak-gateway
   1189622596.126.508+#TMF_Endpoint::Endpoint# salmon
G    1189622596.109.19  trout-gateway
G     1189622596.77.14  yi2250d
/>
```

This is probably the first thing to do after installing Endpoints. If the Endpoint that you attempted to install appears on the output of the wep command, it means the Endpoint installation is complete and it is working fine. If you create a shell script using the wep command to check the successful installation of Endpoints, it can make the installation completion check easy and effective.

If you would like to check for more detailed information about the Endpoint which you installed, you can use the Endpoint Web interface or the wadminep command. They enable you to browse the Endpoint log file. The following shows a sample output of the Endpoint log file (lcfd.log) by the wadminep command.

```
/>wadminep salmon view_log_file
Performing browse mode 'view_log_file' on endpoint 'salmon'
Nov 23 15:41:26 1 lcfd CacheInit: Starting new index file: C:\Tivoli\lcf\dat\1\cache\Index.v5
Nov 23 15:41:26 1 lcfd lcfd 5 (w32-ix86)
Nov 23 15:41:26 1 lcfd Binary Dir (load_dir): 'C:\Tivoli\lcf\bin\w32-ix86\mrt'
Nov 23 15:41:26 1 lcfd Library Dir (lib_dir): 'C:\Tivoli\lcf\bin\w32-ix86\mrt'
Nov 23 15:41:26 1 lcfd Dat Dir (run_dir): 'C:\Tivoli\lcf\dat\1'
Nov 23 15:41:26 1 lcfd Cache Dir (cache_loc): 'C:\Tivoli\lcf\dat\1\cache'
Nov 23 15:41:26 1 lcfd Logging to (logfile): 'C:\Tivoli\lcf\dat\1\lcfd.log' at level 1
Nov 23 15:41:26 1 lcfd Cache limit: '20480000'
Nov 23 15:41:26 1 lcfd Cache size at initialization: '0'
Nov 23 15:41:26 1 lcfd ^Hmm... looks like you're running NT 4.0  (build 1381).  Don't create a console.
Nov 23 15:41:26 1 lcfd node_login: listener addr '0.0.0.0+9494'
Nov 23 15:42:35 1 lcfd salmon is dispatcher 130 in region 1189622596
Nov 23 15:42:35 1 lcfd write login file 'lcf.dat' complete
Nov 23 15:42:35 1 lcfd Logging into new gateway...
Nov 23 15:42:36 1 lcfd salmon is dispatcher 130 in region 1189622596
Nov 23 15:42:36 1 lcfd write login file 'lcf.dat' complete
Nov 23 15:42:36 1 lcfd final pid: 99
Nov 23 15:42:36 1 lcfd Login to gateway 9.3.1.210+9494 complete.
Nov 23 15:42:36 1 lcfd Ready.  Waiting for requests (0.0.0.0+9494). Timeout 120.
Nov 23 15:42:36 1 lcfd Spawning: C:\Tivoli\lcf\dat\1\cache\bin\w32-ix86\endpoint\msg_bind.exe, ses: 248be5cb
Nov 23 15:42:46 1 lcfd Spawning: C:\Tivoli\lcf\dat\1\cache\bin\w32-ix86\endpoint\admin.exe, ses: 248be5cd
Nov 23 15:42:47 1 browse Mode = view_log_file
```

In this output, you can see the following line:

```
Login to Gateway 9.3.1.210+9494 complete.
```

This means the Endpoint logged into the Endpoint gateway without an error.

3.3.2.2 Completion Checking from Endpoint

We introduced how to check for a successful Endpoint installation from an administrator’s workstation. However, how can the user who uses the machine where the Endpoint is installed know if the installation is complete? Users may sometimes want to know about the Endpoint installation completion. For these users, we developed a sample tool that enables the user of the Endpoint to know about the installation status, as we discussed in Chapter 3.3.1.2, “Sample Login Script” on page 66.

To enable the option that notifies the user regarding the completion of the Endpoint login process, you should create two scripts in the directory that contains the login script. By default, this would be the C:\WINNT\system32\Repl\Import\Scripts directory. You also have to copy the following files from the Endpoint tools directory of the TMR Server machine (/usr/local/Tivoli/bin/lcf_bundle/bin/w32-ix86/tools) to the same directory.

- echo.exe
- grep.exe
- ls.exe
- sh.exe
- sleep.exe
- tail.exe
- xargs.exe
- win32gnu.dll

The sample consists of two scripts, and the current version of these scripts only supports the Windows NT Endpoint:

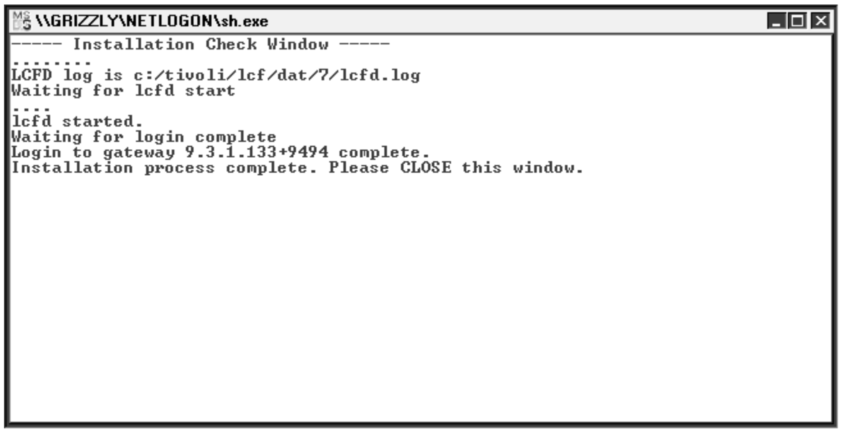
```
#!/bin/sh
#
# logchk - login check script
#
TBIN=//grizzly/netlogon
LOCATION=c:/tivoli/lcf/dat

echo "----- Installation Check Window -----"
while ${TBIN}/sleep 1
do
echo -n "."
CURRENT=`${TBIN}/ls -tr ${LOCATION} 2>/dev/null|${TBIN}/tail -1`
if [ ! "${CURRENT}" = "" ];then
break;
fi
done
LCFDLOG=${LOCATION}/${CURRENT}/lcfd.log
echo ""
echo "LCFD log is ${LCFDLOG}"

echo "Waiting for lcfd start "
while ${TBIN}/sleep 1
do
echo -n "."
if [ -f ${LCFDLOG} ];then
break;
fi
done

echo ""
echo "lcfd started."
echo "Waiting for login complete"

${TBIN}/tail -f -n 1000 ${LCFDLOG} | ${TBIN}/xargs -l1 sh rd
read ch
```

Another script is shown below. This script checks the contents of the lcfd.log file and displays important information on stdout:

```
#!/bin/sh
#
# rd - lcfd.log retrieve script
#
set +x

a_line=$*
set $a_line
LVL=$4
shift;shift;shift;shift;shift
CONTENTS=$*
if [ $LVL != "1" ];then
exit
fi
set $CONTENTS
#
# Level 1 Messages
#
```

```
# Successfully complete
if [ "Login to gateway complete." = "$1 $2 $3 $5" ];then
echo "$1 $2 $3 $4 $5"
echo "Installation process complete. Please CLOSE this window."
read ch
exit
fi
# Login trial to Login Interfaces failed
if [ "No gateway found." = "$a_line" ];then
echo $a_line
exit
fi
# Normal login failure.
if [ "gw login failure:" = "$1 $2 $3" ];then
echo "Failure for Normal Login"
exit
fi

#
# Level 2 Messages
#
# Send login packet
if [ "net_usend of" = "$1 $2" ];then
echo "Try to login $6"
exit
fi
# Login attempt message
if [ "send_login_dgram: waiting for reply. attempt" = "$1 $2 $3 $4 $5" ];then
echo "$5 $6 $7 $8"
exit
fi
# Login trial failed and Entering wait loop.
if [ "Entering net_wait_for_connection," = "$1 $2" ];then
echo "Entering wait loop. $3"
exit
fi
# Normal login attempt to Gateway.
if [ "Connecting to" = "$1 $2" ];then
echo $a_line
exit
fi
```

When the Endpoint installation and the Endpoint login process completes, this sample script displays the Endpoint login process information as follows:



```
\\GRIZZLY\NETLOGON\sh.exe
----- Installation Check Window -----
........
LCFD log is c:/tivoli/lcf/dat/7/lcfd.log
Waiting for lcfd start
....
lcfd started.
Waiting for login complete
Login to gateway 9.3.1.133+9494 complete.
Installation process complete. Please CLOSE this window.
```

Figure 21. Output of logchk script

3.3.3 TMR Redirection

Version 3.6 of Tivoli supports the TMR redirection feature. This feature allows you to configure a special TMR to perform as the master router for the Endpoint login process across many regions (TMRs).

To redirect the TMR, you need to define the select_gateway_policy script that specifies the appropriate Gateways for an Endpoint that has requested an initial login. Normally, the select_gateway_policy script just returns the candidate Gateways in a single TMR for the Endpoint logging in, but you can specify an Endpoint gateway belonging to another TMR. This is called the TMR redirection function.

TMR redirection can simplify the Endpoint configuration process in a large environment or an environment where the broadcast approach is not adequate. If you would not like to use the broadcast for the Endpoint login or the Endpoint login is not completed with the broadcast, some configuration for the Endpoint login might be necessary for completing the Endpoint login. Specifying a specific Gateway increases the effort for Endpoint configuration at initial login and planning for the allocation of resources, the Endpoint Manager or Endpoint gateways. In this situation, specifying the proper Gateway for each Endpoint by Gateway name or IP address by the use of installation options can be complex.

To resolve this problem, you can use the TMR redirection function. In the TMR redirection, the Endpoint gateway (which works as the master router) should be reachable from the all Endpoints, even if the Endpoint belongs to another TMR. If you define the well-known name as the Endpoint gateway host name, such as tivoli-gateway, all Endpoint can specify the well-known name using the -g tivoli-gateway option at the Endpoint initial login. Then the select_gateway_policy assigns each Endpoint to the appropriate Endpoint gateway across multiple TMRs. This TMR redirection makes the Endpoint configuration at the initial login simple and easy.

We will provide more detailed information about TMR redirection in “Configuring the TMA Environment” on page 105.

3.4 Configuring Preloaded TMA

The preloaded TMA will be provided for several platform types in the near future. In the previous chapter, “Tivoli Ready with Tivoli Management Agent” on page 31, we introduced two different types of the preloaded TMA, the zipped TMA code and preloaded TMA in OS (or application). In this section, we introduce how to configure the preloaded TMA for each type on the Windows NT and AIX platform.

3.4.1 Preloaded TMA for Windows NT (Zipped TMA)

As we mentioned, the zipped TMA code will be distributed from the Tivoli Web site (`http://www.tivoli.com`). The size of the whole zip file is approximately 3 MB and preloaded TMA for NT is approximate 600 KB. The following are instructions for how to unpack and activate the zipped TMA code on Windows NT.

1. Download the zip file from the Web site `http://www.tivoli.com`.
2. Locate the zip file in the appropriate directory.
3. Unzip the self-extracting zip file to the appropriate directory. Then, the subdirectories appear under the directory to which the zip file is extracted. Each subdirectory corresponds to the various platforms (refer to Figure 22 on page 74).
4. Select the appropriate subdirectory. In this case, select the winnt subdirectory.
5. Execute the batch file (ntins.bat) in the winnt subdirectory. After executing the batch file, this machine should be a preloaded TMA machine. This means that the `lcfd` daemon has not been started at that time. Therefore, you need to activate the `lcfd` to use the TMA immediately.
6. The zipped TMA is installed under the C:\Tivoli\lcf directory. To activate the Tivoli Authentication Package (TAP), execute the following command:

   ```
   C:\Tivoli\lcf\bin\w32-ix86\mrt\wlcftap.exe -a
   ```

Reboot the machine.

7. To add the Tivoli reserved user, execute the following command:

```
C:\Tivoli\lcf\bin\w32-ix86\mrt\ntconfig.exe -e
```

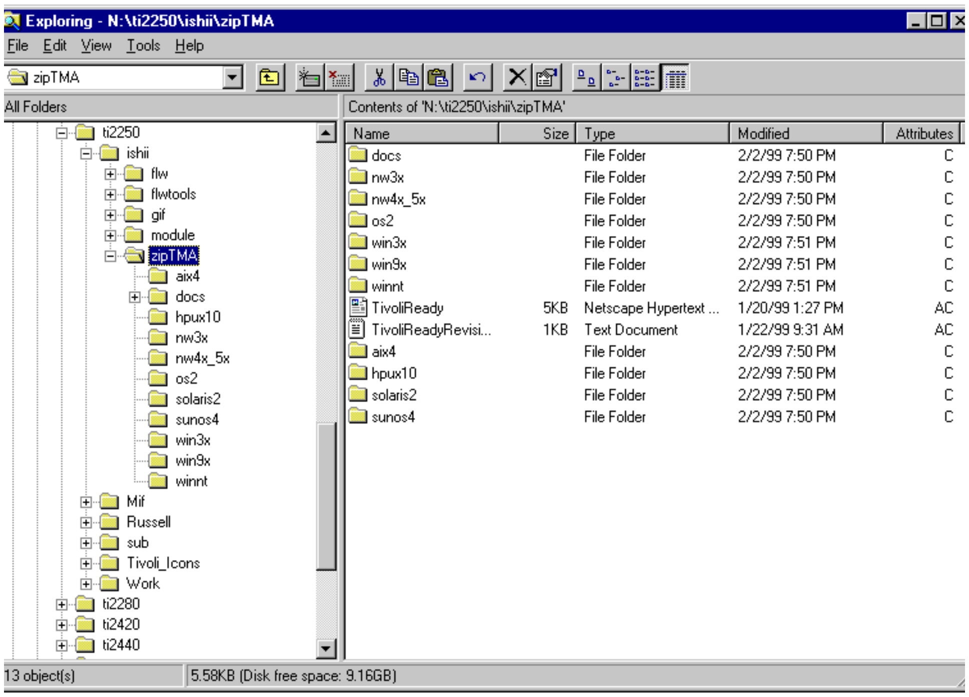
8. To start the `lcfd` daemon, execute the `lcfd` command with the appropriate options. This depends on the environment. The following is an example of the `lcfd` command:

```
C:\Tivoli\lcf\bin\w32-ix86\mrt\lcfd.exe -i -g gw_addr+9494 -D
bcast_disabled=1
```

After this operation, the Tivoli Endpoint is configured as an NT service, so it allows you to have the `lcfd` daemon restart each time the machine is rebooted.

9. If you would like to install the NT taskbar icon, execute the following command (this operation is optional):

```
C:\Tivoli\lcf\bin\w32-ix86\mrt\lcfep.exe -x -i
```



Contents of 'N:\ti2250\ishii\zipTMA'

Name	Size	Type	Modified	Attributes
docs		File Folder	2/2/99 7:50 PM	C
nw3x		File Folder	2/2/99 7:50 PM	C
nw4x_5x		File Folder	2/2/99 7:50 PM	C
os2		File Folder	2/2/99 7:50 PM	C
win3x		File Folder	2/2/99 7:51 PM	C
win9x		File Folder	2/2/99 7:51 PM	C
winnt		File Folder	2/2/99 7:51 PM	C
TivoliReady	5KB	Netscape Hypertext ...	1/20/99 1:27 PM	AC
TivoliReadyRevisi...	1KB	Text Document	1/22/99 9:31 AM	AC
aix4		File Folder	2/2/99 7:50 PM	C
hpux10		File Folder	2/2/99 7:50 PM	C
solaris2		File Folder	2/2/99 7:50 PM	C
sunos4		File Folder	2/2/99 7:50 PM	C

13 object(s) 5.58KB (Disk free space: 9.16GB)

Figure 22. The Zipped TMA Images for Each Platform

Note

If you have unpacked the zipped TMA code on the machine successfully, the following HTML files provide the activation instructions and are located in the C:\Tivoli\TivReady directory. You can refer to it as well during the activation process.

```
C:\Tivoli\TivReady\activate.htm
C:\Tivoli\TivReady\Readme.htm
```

For mass activation, the following sample NT login script is provided by the zipped TMA code.

```
C:\Tivoli\lcf\generic\logontma.bat
```

We will introduce how to activate an Endpoint using the NT login script in the next section. It is a powerful way to deploy masses of preloaded TMA machines and activate them.

3.4.2 Preloaded TMA for Windows NT (Preloaded TMA in OS)

As we mentioned, there are two ways to activate the preloaded TMA, manually or by using an NT login script. This section describes how to use an NT login script to activate the preloaded TMA for Windows NT. As we mentioned, since the Endpoint software has already been installed, the preloaded TMA does not require the installation operation, so the setup programs, such as InstallShield, are not required. The following sample script activates the preloaded TMA for Windows NT:

```
@echo off
REM Login.bat
REM Silently abort if not running Windows
if "%windir%" == "" goto end

if "%OS%" == "Windows_NT" goto NT
REM Silently abort if not running Windows NT
goto end

:NT
if not exist %SystemDrive%\Tivoli\lcf\bin\w32-ix86\mrt\lcfd.exe goto alert

set LCFOPTS=%1 %2 %3 %4 %5 %6 %7 %8 %9
set LCFROOT=%SystemDrive%\Tivoli\lcf

REM Start TAP
%LCFROOT%\bin\w32-ix86\mrt\wlcftap -a

REM Start the TMA and install it as a service
%LCFROOT%\bin\w32-ix86\mrt\lcfd.exe -i -C %LCFROOT%\dat\1 %LCFOPTS%

set LCFOPTS=
set LCFROOT=

goto end

:alert
echo Could not find the files needed to start the Tivoli Management Agent.
echo Please contact your Tivoli Administrator.
goto end

:end
```

The following sample NT login script is provided on the preloaded TMA machine:

```
c:\Tivoli\lcf\generic\logontma.bat
```

3.4.2.1 Activation Instructions

The following shows how to activate the preloaded TMA for Windows NT:

1. On the NT Domain Server, copy the above sample script to the scripts directory. By default, the directory is:

   C:\Winnt\system32\Repl\Import\Scripts.

2. Run the User Manager for Domains utility, select a user and then edit the Profile definition. Enter the path of the script into the Logon Scripts field.
3. Whenever the user logs in to this domain, the sample activation script runs on its machine and activates/configures the preloaded TMA.

The sample script would pass any options of the `lcfd` command to the `lcfd` daemon. Don't put the parameters directly into the Logon Scripts field of the Windows NT User Manager for Domain utility, because it does not support the parameters that are passed to the login script. You should pass the parameters through another batch script as follows, and specify the script name in the Logon Scripts field without the parameter.

```
call login "-g grizzly+9494 -D bcast_disable=1 -d2"
```

You can also run this script stand-alone.

3.4.3 Preloaded TMA for UNIX (Zipped TMA Code)

The zipped TMA code can be obtained from the Tivoli Web site. The zip file includes the preloaded TMA image for AIX platform as well. The preloaded TMA for AIX is approximately 600 KB. The following are instructions of how to unpack and activate the zipped TMA code on AIX.

1. Download the zip file from the Tivoli Web site.
2. Locate the zip file in the appropriate directory.
3. Unzip the zip file to the appropriate directory using an unzip utility. The subdirectories appear under the directory to which the zip file is extracted.
4. Select the aix4 su-directory and copy the aix4-r1tma.tar file to the root directory (/)on the AIX machine. This tar file contains the following files:

```
# tar -tvf aix4-r1.tar
drwxr-xr-x 395 40      0 Jan 15 15:29:13 1999 etc/
drwxr-xr-x 395 40      0 Jan 15 15:29:13 1999 etc/Tivoli/
drwxr-xr-x 395 40      0 Jan 15 15:29:13 1999 etc/Tivoli/lcf/
drwxr-xr-x 395 40      0 Jan 15 15:29:13 1999 etc/Tivoli/lcf/1/
-rw-r--r-- 395 40   2953 Sep 25 10:04:54 1998 etc/Tivoli/lcf/1/lcf_env.csh
-rw-r--r-- 395 40   1704 Sep 25 10:05:06 1998 etc/Tivoli/lcf/1/lcf_env.sh
drwxr-xr-x 395 40      0 Jan 15 15:29:13 1999 opt/
drwxr-xr-x 395 40      0 Jan 15 15:29:15 1999 opt/Tivoli/
drwxr-xr-x 395 40      0 Jan 15 15:29:14 1999 opt/Tivoli/lcf/
drwxr-xr-x 395 40      0 Jan 15 15:29:14 1999 opt/Tivoli/lcf/bin/
drwxr-xr-x 395 40      0 Jan 15 15:29:14 1999 opt/Tivoli/lcf/bin/aix4-r1/
drwxr-xr-x 395 40      0 Jan 15 15:29:14 1999 opt/Tivoli/lcf/bin/aix4-r1/mrt/
-rw-r--r-- 395 40  95929 Aug 10 20:31:28 1998 opt/Tivoli/lcf/bin/aix4-r1/mrt/lcfd
drwxr-xr-x 395 40      0 Jan 15 15:29:14 1999 opt/Tivoli/lcf/dat/
drwxr-xr-x 395 40      0 Jan 15 15:29:14 1999 opt/Tivoli/lcf/dat/1/
-rw-r--r-- 395 40   4246 Dec 20 19:19:52 1998 opt/Tivoli/lcf/dat/1/lcfd.sh
drwxr-xr-x 395 40      0 Jan 15 15:29:14 1999 opt/Tivoli/lcf/generic/
-rw-r--r-- 395 40   4874 Oct 06 08:48:44 1998 opt/Tivoli/lcf/generic/as.sh
-rw-r--r-- 395 40    391 Dec 20 19:36:23 1998 opt/Tivoli/lcf/generic/readme
drwxr-xr-x 395 40      0 Jan 15 15:29:14 1999 opt/Tivoli/lcf/lib/
drwxr-xr-x 395 40      0 Jan 15 15:29:14 1999 opt/Tivoli/lcf/lib/aix4-r1/
-rw-r--r-- 395 40  22881 Aug 10 20:31:12 1998 opt/Tivoli/lcf/lib/aix4-r1/libcpl.a
-rw-r--r-- 395 40  43280 Aug 10 20:31:12 1998 opt/Tivoli/lcf/lib/aix4-r1/libdes.a
-rw-r--r-- 395 40 442711 Aug 10 20:31:16 1998 opt/Tivoli/lcf/lib/aix4-r1/libmrt.a
drwxr-xr-x 395 40      0 Jan 15 15:29:16 1999 opt/Tivoli/TivoliReady/
-rw-r--r-- 395 40   3875 Jan 15 10:07:16 1999 opt/Tivoli/TivoliReady/activate.html
drwxr-xr-x 395 40      0 Jan 15 15:29:16 1999 opt/Tivoli/TivoliReady/html/
-rw-r--r-- 395 40   2672 Sep 23 15:46:35 1998 opt/Tivoli/TivoliReady/html/bkgrd3.gif
-rw-r--r-- 395 40   1122 Oct 14 05:20:54 1998 opt/Tivoli/TivoliReady/html/bottom_bar.gif
-rw-r--r-- 395 40   5603 Sep 23 15:46:35 1998 opt/Tivoli/TivoliReady/html/header.gif
-rw-r--r-- 395 40   2034 Sep 23 15:46:38 1998 opt/Tivoli/TivoliReady/html/tivrdy.gif
-rw-r--r-- 395 40   1203 Sep 23 15:46:35 1998 opt/Tivoli/TivoliReady/html/top_bar_index.gif
-rw-r--r-- 395 40     43 Oct 14 04:44:34 1998 opt/Tivoli/TivoliReady/html/white_dot.gif
-rw-r--r-- 395 40   4119 Dec 20 23:49:04 1998 opt/Tivoli/TivoliReady/Readme.html
#
```

5. Untar the aix4-r1tma.tar file in the root directory using the following command:

   `tar xvf aix4-r1tma.tar`

   Then, the preloaded TMA code is extracted and this AIX machine will be a preloaded TMA machine. As you know, the `lcfd` daemon has not been started yet at that time, so that you need to activate the `lcfd` daemon.

6. Change the directory to the /opt/Tivoli/lcf/generic directory:

   `cd /opt/Tivoli/lcf/generic`

7. Execute the autostart script with the appropriate option in the directory as follows:

```
# sh /as.sh 1
Performing auto start configuration
#
```

   Then the following entry is added into the /etc/inittab file:

```
rctma1:2:wait:/etc/rc.tma1 > /dev/console 2>&1 # Tivoli Management Agent
```

The `/etc/rc.tma1` file is also created for the auto start configuration. The following are the contents of the `/etc/rc.tma1` file:

```
#!/bin/sh
#
#              Start the Tivoli Management Agent
#
if [ -f /opt/Tivoli/lcf/dat/1/lcfd.sh ]; then
    /opt/Tivoli/lcf/dat/1/lcfd.sh start
fi
```

After this operation, you can have the `lcfd` daemon restart each time the machine is rebooted.

Note

In our test environment, we needed to modify some files to configure the autostart properly. In the as.sh script, you may need to modify the `ETRC` environment variable as follows:

```
ETRC=/opt/Tivoli/lcf/dat/1/lcfd.sh
```

By default, the `lcfd.sh` and `lcfd` command did not have execute permission, so you may need to execute the following commands:

```
chmod +x /opt/Tivoli/lcf/dat/1/lcfd.sh
chmod +x /opt/Tivoli/lcf/bin/aix4-r1/mrt/lcfd
```

The zipped TMA code which we used was not generally available code. It could depend on which version of the zipped TMA you use, so that you should check the above items when you use the zipped TMA code.

3.4.3.1 Specifying Endpoint Options

In the activation process, to specify the Endpoint options, you need to define the `LCFOPTS` environment variable as follows:

```
LCFOPTS='-g gw_addr+9494 -D bcast_disabled=1'
export LCFOPTS
```

The `LCFOPTS` environment variable is used in the lcfd.sh script each time the `lcfd` daemon starts.

To perform the mass activation, you need to write a shell script that executes the as.sh script remotely using the `rexec` or `rsh` feature. By default, the as.sh script starts the `lcfd` daemon with the default options. This means that the `lcfd` daemon broadcasts the login request immediately. To specify the appropriate Endpoint options, your shell script should include the process which enables the `LCFOPTS` environment variable in each system.

Note

If you have untarred the tar file which contains the preloaded TMA for AIX code on the AIX machine successfully, the following HTML and readme files which provide the activation and auto start configuration instructions are located in the /opt/Tivoli/TivoliReady and /opt/Tivoli/lcf/generic directory. You can refer to it as well during the activation process.

```
/opt/Tivoli/TivoliReady/activate.html
/opt/Tivoli/TivoliReady/Readme.html
/opt/Tivoli/lcf/generic/readme
```

3.4.4 Preloaded TMA for AIX (Preloaded TMA in OS)

The preloaded TMA in AIX is a little different. The preloaded TMA in AIX is shipped as the installp format and it is included in the AIX installation CD-ROM, so that you can handle it as one of the AIX LPPs (Licensed Program Product). We introduce what is preloaded of TMA on AIX and how to activate it.

3.4.4.1 What Is Preloaded TMA on AIX?

As we mentioned, the preloaded TMA on AIX is shipped as part of the AIX installation CD-ROM. In other words, the preloaded TMA in AIX is shipped as a fileset. The fileset name of the preloaded TMA for AIX is:

```
    Tivoli_Management_Agent.client.rte
```

This means you can use the `lslpp` command for installing and uninstalling the software or browsing the software information. The following is the output of the `lslpp` command and shows the files which are shipped as the preloaded TMA in AIX.

```
# lslpp -f Tivoli_Management_Agent.client.rte
  Fileset               File
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  Tivoli_Management_Agent.client.rte 3.2.0.0
                        /usr/lpp/Tivoli_Management_Agent.client/bin/autostart.sh
                        /usr/lpp/Tivoli_Management_Agent.client
                        /usr/lpp/Tivoli_Management_Agent.client/productid
                        /usr/lpp/Tivoli_Management_Agent.client/bin
                        /usr/lpp/Tivoli_Management_Agent.client/Readme.txt

Path: /etc/objrepos
  Tivoli_Management_Agent.client.rte 3.2.0.0
                        /opt/Tivoli/lcf/dat/1/lcfd.sh
                        /opt/Tivoli/lcf/bin/aix4-r1
                        /opt/Tivoli/lcf/dat
                        /opt/Tivoli/lcf/lib/aix4-r1
                        /opt/Tivoli/lcf/lib/aix4-r1/libcpl.a
                        /opt/Tivoli/lcf/bin
                        /opt/Tivoli/lcf/lib/aix4-r1/libmrt.a
                        /opt
                        /opt/Tivoli
                        /opt/Tivoli/lcf/lib/aix4-r1/libdes.a
                        /opt/Tivoli/lcf
                        /opt/Tivoli/lcf/bin/aix4-r1/mrt/lcfd
                        /opt/Tivoli/lcf/lib/aix4-r1/libccms_lcf.a
                        /opt/Tivoli/lcf/bin/aix4-r1/mrt
                        /opt/Tivoli/lcf/dat/1
#
```

Therefore, the above files are installed onto the AIX machine as the preloaded TMA.

3.4.4.2 Activation Instruction

The preloaded TMA for AIX is in the installp format image. To activate it, you need to install the installp image first, then invoke the shell script which activates the Endpoint process.

The following steps show how to activate the preloaded TMA for AIX:

1. Install the `Tivoli_Management_Agent.client.rte` fileset to the AIX machine using smit. From the smit menu, select the following menu:

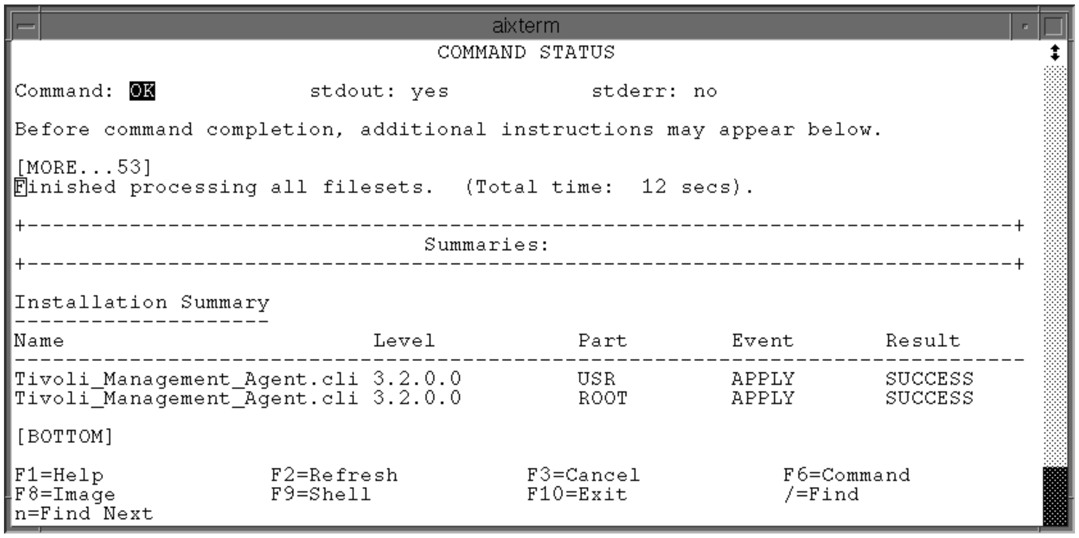
   ```
   # smit install
     -> Install and Update Software
       -> Install and Update from ALL Available Software
         -> INPUT device / directory for software
           -> SOFTWARE to install
   ```

   Then press the **PF4** key to browse the available software on the CD-ROM. The following panel appears:

```
SOFTWARE to install

Move cursor to desired item and press F7. Use arrow keys to scroll.
    ONE OR MORE items can be selected.
Press Enter AFTER making all selections.

[MORE...11]
   + 5.3.0.0  TotalNET Advanced Server

  Tivoli_Management_Agent.client                                     ALL
   + 3.2.0.0  Management Agent runtime"

  adsm.afs.client.aix42                                              ALL
   + 3.1.20.3  ADSM Client - AFS File Backup Client

  adsm.api.client.aix42                                              ALL
[MORE...161]

F1=Help                 F2=Refresh              F3=Cancel
F7=Select               F8=Image                F10=Exit
Enter=Do                /=Find                  n=Find Next
```

From the panel, select the **Tivoli Management Agent client software** with the PF7 key and install it.

2. When the installation process completes, the following messages appear on the smit panel:



```
                                COMMAND STATUS

Command: OK            stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

[MORE...53]
Finished processing all filesets.  (Total time:  12 secs).

+-----------------------------------------------------------------------------+
                               Summaries:
+-----------------------------------------------------------------------------+

Installation Summary
--------------------
Name                         Level           Part        Event       Result
-------------------------------------------------------------------------------
Tivoli_Management_Agent.cli 3.2.0.0          USR         APPLY       SUCCESS
Tivoli_Management_Agent.cli 3.2.0.0          ROOT        APPLY       SUCCESS

[BOTTOM]

F1=Help             F2=Refresh          F3=Cancel           F6=Command
F8=Image            F9=Shell            F10=Exit            /=Find
n=Find Next
```

Figure 23. Preloaded TMA for AIX Installation Completion

To confirm the installation, you can also use the `lslpp` command as follows:

```
# lslpp -ah Tivoli_Management_Agent.client.rte
  Fileset         Level     Action       Status       Date         Time
  ---------------------------------------------------------------------------
Path: /usr/lib/objrepos
  Tivoli_Management_Agent.client.rte
                  3.2.0.0   COMMIT       COMPLETE     01/21/99     09:41:51
                  3.2.0.0   APPLY        COMPLETE     01/21/99     09:41:51

Path: /etc/objrepos
  Tivoli_Management_Agent.client.rte
                  3.2.0.0   COMMIT       COMPLETE     01/21/99     09:42:03
                  3.2.0.0   APPLY        COMPLETE     01/21/99     09:42:03
#
```

3. At this time, the Endpoint process (`lcfd`) has not been started yet, so you have to activate the preloaded TMA for AIX. The preloaded TMA for AIX provides the autostart script (autostart.sh) and it enables you to have the Endpoint process (`lcfd`) restart each time the machine is rebooted.

   To activate the preloaded TMA in AIX, invoke the autostart.sh script, the /etc/inittab file is modified and the `lcfd` daemon will be started and restarted each time the machine rebooted. This process is the same as the zipped TMA code.

3.5 Using Installation Tools

In this section, we introduce some tools that can be useful for Endpoint installation.

3.5.1 Using Silent Installation

There are two ways to configure the silent installation for the Windows operating system: one is using the provided (default) configuration file and another is creating a new configuration file. Normally, the configuration file of the silent installation is named setup.iss by default. If the product provides the setup.iss file in the install images, you can modify it directly in order to configure your silent installation. In Version 3.6 of the Tivoli Management Framework CD-ROM, the Endpoint software provides the setup.iss file. If you want to change a small part of the installation defaults, you can modify this file.

If the product does not provide the setup.iss file, or if you want to change the sequence of the installation process, you need to create a new setup.iss file. Since this is normal operation for products which use the InstallShield as the installation method, you can use this same method to create a new setup.iss file for other products.

The following are the step-by-step instructions for this process in a Windows NT Version 4.0 environment.

1. Insert Version 3.6 of the Tivoli Management Framework CD-ROM into the CD-ROM drive of the system. In this case, the CD-ROM drive is configured as the D: drive.
2. From the Start menu, select the **Run** option to display the Run dialog.
3. Enter `d:\pc\lcf\winnt\setup -r -f1c:\temp\setup.iss` in the Open field. The c:\temp\setup.iss is the name of the new setup.iss file that you create. Press **OK** to run the setup program.

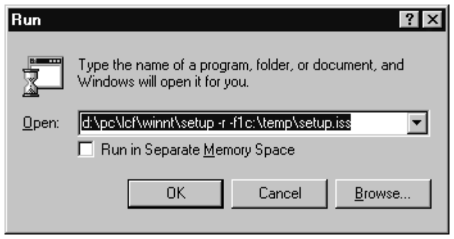
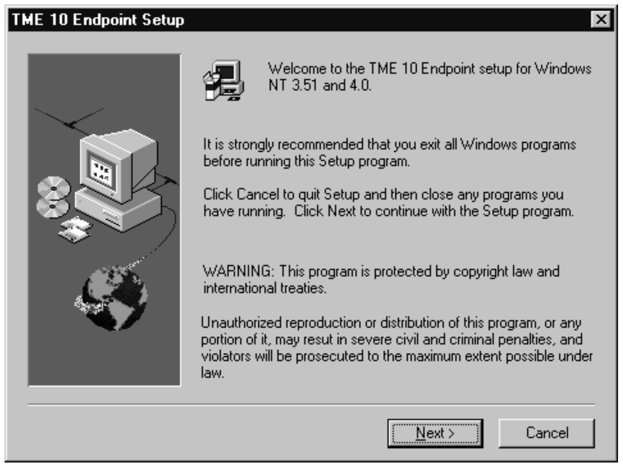

Run
?
Type the name of a program, folder, or document, and Windows will open it for you.
Open:
d:\pc\lcf\winnt\setup -r -f1c:\temp\setup.iss
Run in Separate Memory Space
OK
Cancel
Browse...

Figure 24. Starting InstallShield

4. The InstallShield shows the TME 10 Endpoint Setup dialog.


TME 10 Endpoint Setup

Welcome to the TME 10 Endpoint setup for Windows NT 3.51 and 4.0.

It is strongly recommended that you exit all Windows programs before running this Setup program.

Click Cancel to quit Setup and then close any programs you have running. Click Next to continue with the Setup program.

WARNING: This program is protected by copyright law and international treaties.

Unauthorized reproduction or distribution of this program, or any portion of it, may resut in severe civil and criminal penalties, and violators will be prosecuted to the maximum extent possible under law.

Figure 25. Setup Dialog

5. Press the **Next** button to display the Endpoint Installation option dialog.

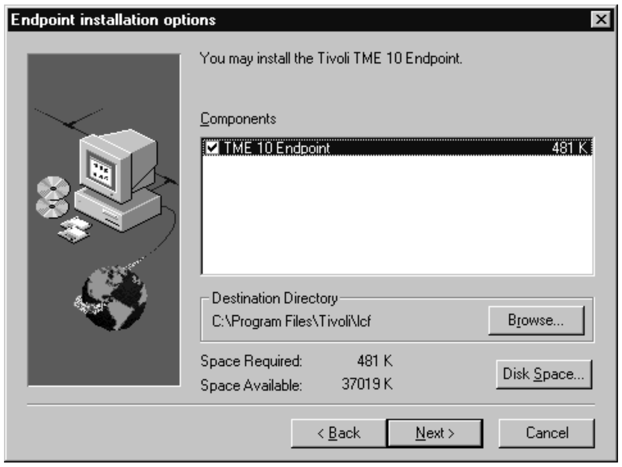


Figure 26. Installation Option Dialog

6. Press the **Browse** button to change the destination directory.

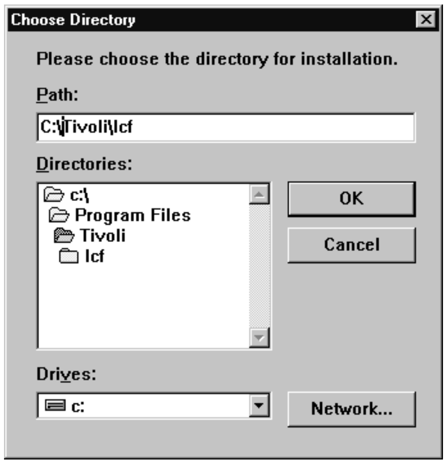
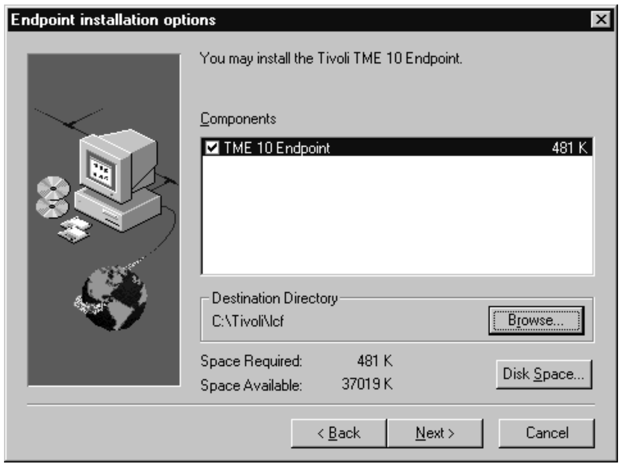


Figure 27. Change Directory Dialog

7. Enter the path name of the destination directory and press **OK**.


Endpoint installation options

You may install the Tivoli TME 10 Endpoint.

Components

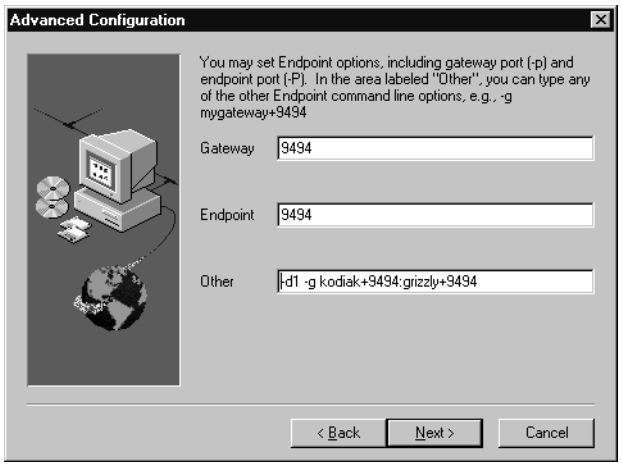
☑ TME 10 Endpoint 481 K

Destination Directory
C:\Tivoli\lcf

Space Required: 481 K
Space Available: 37019 K

Figure 28. Installation Option Dialog

8. Press the **Next** button.


Advanced Configuration

You may set Endpoint options, including gateway port (-p) and endpoint port (-P). In the area labeled "Other", you can type any of the other Endpoint command line options, e.g., -g mygateway+9494

Gateway: 9494

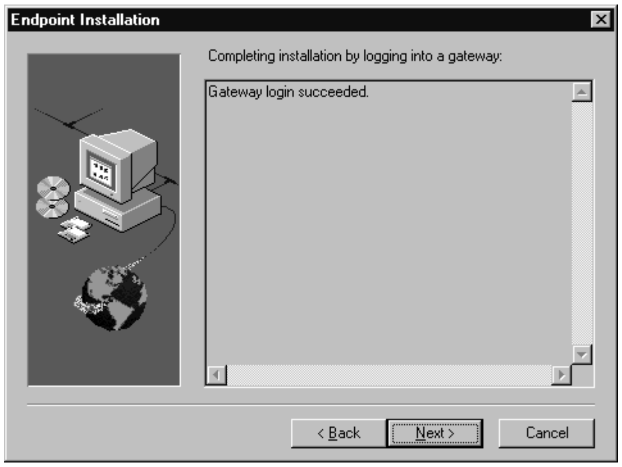
Endpoint: 9494

Other: -d1 -g kodiak+9494:grizzly+9494

Figure 29. Advanced Configuration Dialog

You can specify the `lcfd` daemon options in the Other field. Press the **Next** button to start the Endpoint login process.

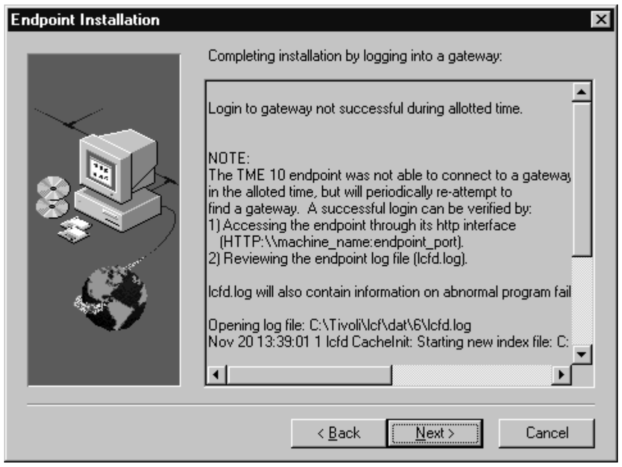
9. When the Endpoint login is completed, the following dialog is displayed.



Gateway login succeeded.

Figure 30. Successfully Installed

10.You may see the following dialog at this time. It is caused by a kind of timeout that is detected by the installer, it does not mean the Endpoint login necessarily failed. You can then press the **Next** button. The Endpoint login process is being done in the background. But you don't need to wait for login completion. This operation just records your responses to create the setup.iss file.

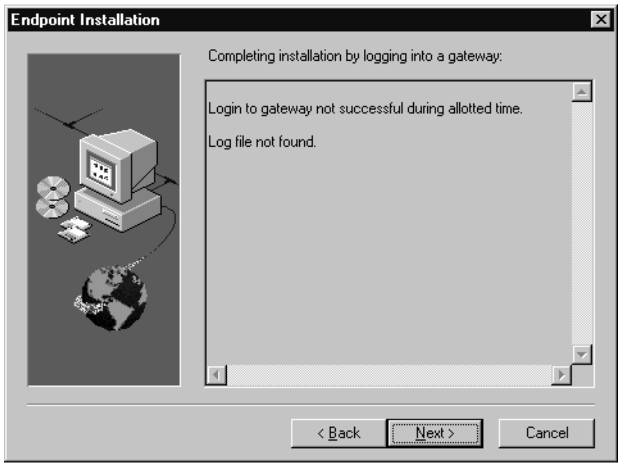


```
Login to gateway not successful during allotted time.


NOTE:
The TME 10 endpoint was not able to connect to a gatewa
in the alloted time, but will periodically re-attempt to
find a gateway.  A successful login can be verified by:
1) Accessing the endpoint through its http interface
   (HTTP:\\machine_name:endpoint_port).
2) Reviewing the endpoint log file (lcfd.log).

lcfd.log will also contain information on abnormal program fail

Opening log file: C:\Tivoli\lcf\dat\6\lcfd.log
Nov 20 13:39:01 1 lcfd CacheInit: Starting new index file: C:
```

Figure 31. Installation Timeout

DRAFT

11.If you see the following dialog, the installation may have failed. In this case, it could be a mistake in the options you specified, so that you should not continue.

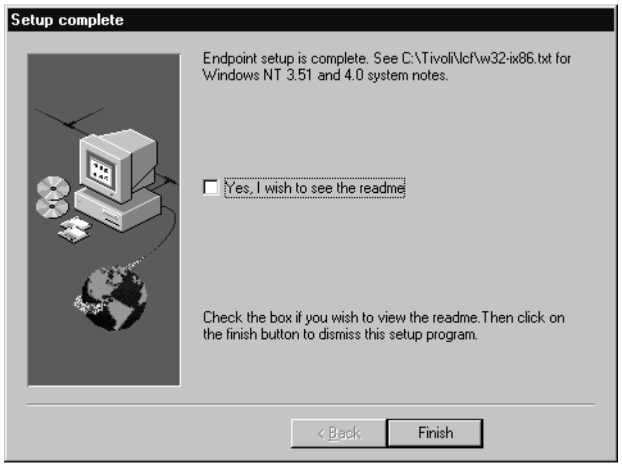


Completing installation by logging into a gateway:

```
Login to gateway not successful during allotted time.
Log file not found.
```

Figure 32. Installation Failure

12.When the Endpoint installation is completed the following dialog is displayed. Then press the **Finish** button.



Endpoint setup is complete. See C:\Tivoli\lcf\w32-ix86.txt for Windows NT 3.51 and 4.0 system notes.

☐ Yes, I wish to see the readme

Check the box if you wish to view the readme.Then click on the finish button to dismiss this setup program.

Figure 33. Setup Complete Dialog

You can find the new `setup.iss` file in the specified directory.

Note

Normally, this type of the setup operation does not install the program module; it just creates the setup.iss file. However, in the case of the Endpoint software, the Endpoint installation and the activation of the Endpoint are actually done. Therefore, when you just want to create the new setup.iss file and do not want to install the Endpoint software to the machine, you have to remove the Endpoint module from the machine after you create the new setup.iss file.

3.5.2 Using TMASERV

The Tivoli Migration Toolkit provides the tool TMASERV that simplifies the Endpoint installation. In this section, we introduce the TMASERV and also provide information about how to use and configure it.

3.5.2.1 What is TMASERV

TMASERV (Tivoli Management Agent Login Service) provides a service that helps you to automate the Endpoint installation to Windows machines when they log into a Windows NT domain. TMASERV can install the Endpoint onto Windows 3.x, Windows 95, Windows 98 and Windows NT systems. Basically, the TMASERV uses the NT login script feature, but it is more sophisticated than the normal NT login script. The following are the advantages of the TMASERV:

- You can specify the host name on which the Endpoint will be installed in the configuration file before the installation.
- You can specify the Endpoint options in the configuration file before the installation.

The TMASERV consists of the TMASERV server, the TMR Server and the Windows NT Primary Domain Controller.

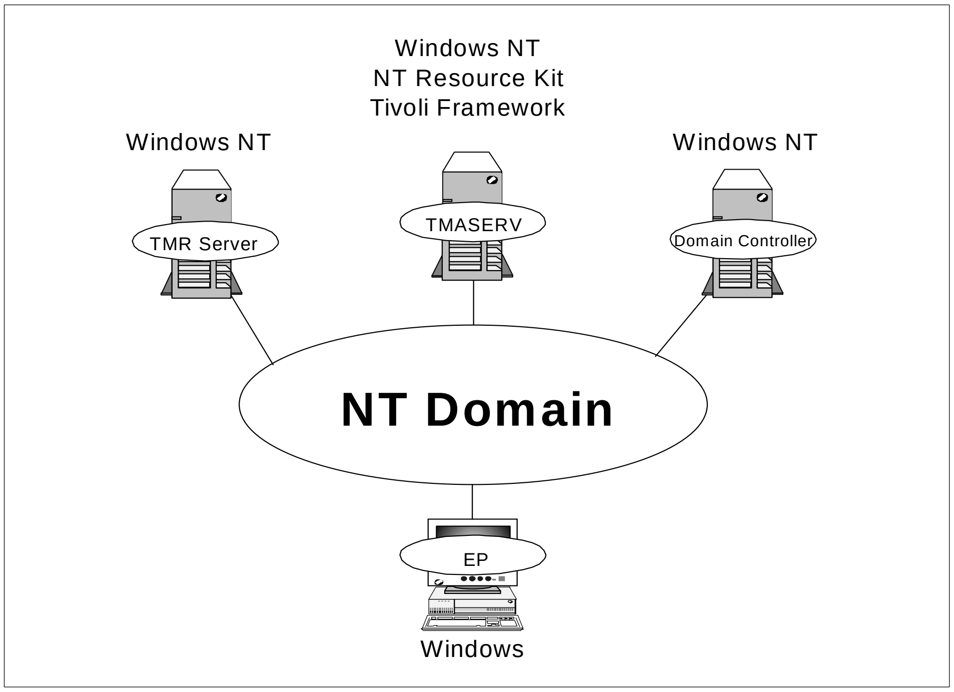


Figure 34. TMASERV Server Environment

The TMASERV has some prerequisites as follows:

- The TMR Server running on Windows NT.
- The Windows NT server machine has the TMASERV software installed. The Windows NT Resource Kit and Version 3.6 of the Tivoli Framework should be installed on this machine.
- The Windows NT Primary Domain Controller to use the NT login script.

The TMASERV requires a Windows NT TMR Server. Using a Windows NT TMR Server is not a problem, but if you are using an AIX TMR Server, you need to do a more complicated customization.

This scenario requires following three files, which are installed only on the TMR Server running Windows NT. You should get them before starting. These are not installed on AIX:

```
C:\Tivoli\bin\lcf_bundle\bin\w32-ix86\deploy\dep_aft.exe
C:\Tivoli\bin\lcf_bundle\bin\win95\deploy\dep_aft.exe
C:\Tivoli\bin\lcf_bundle\bin\win31\deploy\dep_aft.exe
```

Note that TMASERV itself should be run on Windows NT.

3.5.2.2 Using TMASERV Service

To install and start the TMASERV service that installs Endpoints during the login process to the NT domain, follow these steps.

1. Log into the Windows NT system which becomes the TMASERV server because the user with the Administrator authority has the sufficient Tivoli roles to invoke the Tivoli IDL calls. This can be any NT system in your domain. Confirm that the Tivoli Framework is installed and working and the Windows NT Resource Kit is installed. Also confirm that your administrative NT user is registered to the TMR as the Administrator.
2. Setup the Tivoli environment using the `setup_env` command:

   ```
   C:\winnt\system32\drivers\etc\tivoli\setup_env
   ```

3. Copy three files from the \NTLOGON directory in the Tivoli Migration Toolkit CD-ROM. In this case, the CD-ROM drive is configured as the D: drive and the target directory is C:\tivoli\bin\w32-ix86\tools\tmaserv. Do not use a directory name that contains a space such as C:\Program Files.

   ```
   md C:\tivoli\bin\w32-ix86\tools\tmaserv
   cd C:\tivoli\bin\w32-ix86\tools\tmaserv
   copy D:\ntlogin\getpsw.exe
   copy D:\ntlogin\loaas.exe
   copy D:\ntlogin\tmaserv.pl
   ```

4. This operation is required only when you are using the AIX TMR Server. Copy the following files from any Windows NT TMR Server. Note that source means the Windows NT TMR Server and the target means the TMASERV server.

   Source:
   `C:\Tivoli\bin\lcf_bundle\bin\w32-ix86\deploy\dep_aft.exe`

   Target:
   `C:\Tivoli\bin\w32-ix86\tools\tmaserv\bundles\w32-ix86\dep_aft.exe`

   Source:
   `C:\Tivoli\bin\lcf_bundle\bin\win95\deploy\dep_aft.exe`

   Target:
   `C:\Tivoli\bin\w32-ix86\tools\tmaserv\bundles\win95\dep_aft.exe`

   Source:
   `C:\Tivoli\bin\lcf_bundle\bin\win3x\deploy\dep_aft.exe`

   Target:
   `C:\Tivoli\bin\w32-ix86\tools\tmaserv\bundles\win3x\dep_aft.exe`

Modify the `tmaserv.pl` script as follows:

- Open `tmaserv.pl` scripts with an editor.
- Search for the word `&wTransferFile`, and find the following line.

  ```
  $x = &wTransferFile($source, "$BundleDirectory/$interp");
  ```

- Use the `#` character to comment it out as follows:

  ```
  # $x = &wTransferFile($source, "$BundleDirectory/$interp");
  ```

  This line transfers some files from the Windows NT TMR Server, so it is only for a Windows NT TMR Server.

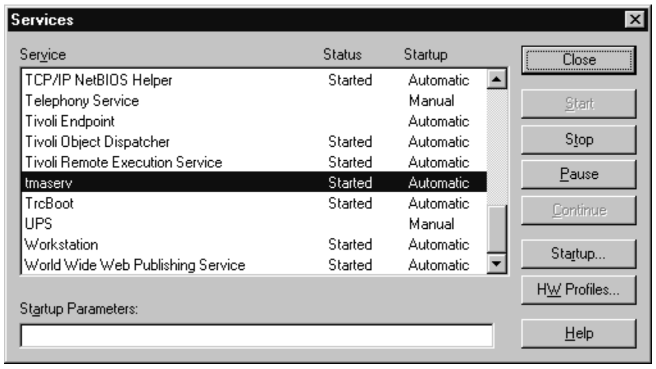
5. Install the TMASERV service using the following command:

   ```
   perl tmaserv.pl -install
   ```

6. Then the install program asks you the user ID and password. Enter the user ID and password of the user account that has administrator authority.
7. When the installation is successful, it automatically starts the TMASERV service and displays the following message:

   ```
   Service is running.
   ```

8. The installation program configures the TMASERV service to start the service automatically each time the server is rebooted. The Windows NT service `tmaserv` is added as follows.



Service	Status	Startup
TCP/IP NetBIOS Helper	Started	Automatic
Telephony Service		Manual
Tivoli Endpoint		Automatic
Tivoli Object Dispatcher	Started	Automatic
Tivoli Remote Execution Service	Started	Automatic
tmaserv	Started	Automatic
TrcBoot	Started	Automatic
UPS		Manual
Workstation	Started	Automatic
World Wide Web Publishing Service	Started	Automatic

Figure 35. Windows NT Services Dialog

3.5.2.3 Configuring TMASERV Service

To use the TMASERV service efficiently, you need to configure the TMASERV service.

Configuring SCANDIR Directory

As we mentioned, the TMASERV service can define the host name which you want to install the Endpoint. It has the advantage of the TMASERV and better than normal NT login script. To define the host name, you need to create a file in the SCANDIR directory that has the same name as the host name of the system on which you want to install the Endpoint. The contents of this file are ignored. For example, if you want to install the Endpoint on the system named `kodiak`, issue the following command from the directory into which you copied the tmaserv.pl file. In our environment, it is the C:\Tivoli\biw32-ix86\tools\tmaserv directory:

```
echo > SCANDIR\kodiak
```

This creates the file kodiak in the SCANDIR subdirectory. The TMASERV service checks the file in this directory every five seconds and if the TMASERV service finds any file in the directory, it starts to install the Endpoint on the system that has the same host name as the file name in the directory.

Configuring lcf_seed File

After the TMASERV service is installed, you can find the following file in the BUNDLES subdirectory.

```
C:\tivoli\bin\w32-ix86\tools\tmaserv\BUNDLES\lcf_seed
```

This is the configuration file for TMASERV service. You should change at least two entries as follows:

```
base_dir=C:\Tivoli\lcf
lcf_opt=-d1 -g kodiak+9494:grizly+9494 -D bcast_disable=1
```

The `lcf_opt` specifies the options for the `lcfd` daemon, so you can use the same options as the `lcfd` command. In this example, the Endpoint (`lcfd`) options are defined as follows:

Trace Level	Level 1
Endpoint Gateways	kodiak and grizzly
Broadcast	Disabled

Please refer to the *Tivoli Framework Migration Guide* for more detail.

3.5.2.4 Creating TMALOGIN.BAT Login Script for Users

In the TMASERV service, the TMALOGIN.BAT script runs when the Windows user logs into the NT domain. If the system is not already an Endpoint, it installs the Endpoint software to the machine and starts it. Place this file on the NT Primary Domain Controller. It is run by any machine that connects to the NT domain.

To install the TMALOGIN.BAT, follow these steps:

1. Log into the NT Primary Domain Controller as the user that has Administrator authority.
2. Copy the TMALOGIN.BAT file from the \NTLOGIN directory in the Tivoli Migration Toolkit CD-ROM to the %SystemRoot%\system32\repl\import\scripts directory (typically C:\Winnt\system32\repl\import\scripts) on the NT Domain Controller.
3. To set the SERVER variable in the `TMALOGIN.BAT` file to the name of the server that you installed the TMASERV on, edit the file, and set the variable as follows:

   ```
   set SERVER=yi2250d
   ```

4. Set the `DESTDIR` variable to the installation location of the Endpoint software as follows.

   ```
   set DESTDIR=C:\Tivoli\lcf
   ```

5. Open the User Manager utility dialog.
6. For each user that should use the login service, follow these steps. Note that all users should have the Administrator authority.
   - Double-click on the name in the list of users to open the User Properties dialog.
   - From the User Properties dialog, click the **Profile** button.
   - From the User Environment Profile dialog, enter the **`tmalogin.bat`** script as the Login script name.
   - Click **OK**.
   - The next time one of these users logs in, the TMALOGIN.BAT script is invoked.

3.6 Error Messages

Some error messages associated with Endpoint login failures are cryptic. In this section, we take a look at some of these messages.

3.6.1 Deleted Endpoint

After having logged in once, the Endpoint knows its region identity from the lcf.dat file. This file contains, among other things, the dispatcher number, which identifies it to its assigned Gateway. When the Endpoint is deleted from the region using the `wdelep` command, the product does not remove the TMA installation on the Endpoint. In particular, the lcf.dat file remains and it now contains an invalid dispatcher number.

If, following the deletion, the Endpoint attempts to re-login to the region using this dispatcher number, the ‘resource not found’ exception will be generated at the Endpoint Manager and also logged at the Endpoint gateway and the Endpoint. In this case, the following messages appear in the gatelog file:

```
1998/09/04 15:59:49 +06: process_node_login: unwrap_login_info (migrate) failed.
Aborting login for dispatcher 2
       Exception message is: Fri Sep 04 15:59:49 1998 (4): resource ‘’ not found
```

3.6.2 Dispatcher Number Conflict

When the message within the TMA architecture fails to satisfy a test of authenticity, you will see the message like this in the gatelog file:

```
1998/09/04 16:11:56 +06: process_node_login: unwrap_login_info failed.
Aborting login for dispatcher 3
       Exception message is: decrypt_data: HMAC does not match encrypted data!
```

This most likely indicates that the sender and the receiver have incompatible security keys. There are some legitimate circumstances in which this can occur, especially in the prototypical environments in which the product is being evaluated or various implementation schemes are being tested. One such circumstance is the dispatcher conflict.

When the message passes from the Endpoint to the Endpoint gateway, the Endpoint gateway identifies the Endpoint by the dispatcher number in a cleartext header. It maps this to the key that was generated at the Endpoint login. If these don’t match at the Endpoint and Endpoint gateway, the message authentication will fail.

Now, suppose you build up a test TMR and a few Endpoints are installed and login to the TMR. Later, you decide to replace the TMR for whatever reason, but the old Endpoints remain where you have installed them. Then, you create some additional Endpoints in the new TMR. Since it is a new TMR, the dispatcher numbers start over and one of your new Endpoints may be using the same number as the old one. When the old Endpoint attempts to log in, the Endpoint gateway will think it recognizes the Endpoint by the dispatcher. However, the key won’t match, hence, the HMAC failure.

Another odnum (Object Dispatcher Number) conflict scenario can arise from a Tivoli object database restore. The Tivoli database restore may change the next available odnum such that the Endpoint logging in subsequent to the restore can receive an odnum which is already assigned to an old Endpoint. The old Endpoint will see the HMAC failure when trying to log in.

As we mentioned, this message, HMAC error, is a common error in TMA configurations. Especially in new environments where you may be installing and reinstalling Endpoints. In the project that resulted in this redbook, we saw this problem often in our test lab. We usually were able to fix the problem by carrying out the following steps:

1. Stop the `lcfd` daemon.
2. Delete the entry of the Endpoint using the `wdelep` command.
3. Remove the lcf.dat file in the Endpoint.
4. Restart the `lcfd` daemon with the appropriate the options.

This will change the odnum of the Endpoint so be careful.

3.7 TMA Implementation Considerations

In the TMA environment, as of the writing of this redbook, there are some restrictions and limitations. To implement or deploy TMA successfully, the administrators should know about these restrictions. In any case, the administrator should refer to the latest *Tivoli Framework Release Notes* when installing TMA. This section covers some of the more common restrictions and known bugs that will be resolved in future releases.

3.7.1 Installation Considerations

In this section, we introduce various considerations for Endpoint installation and operation.

3.7.1.1 Endpoint Naming Syntax

There are different default naming syntaxes used for Endpoints and it depends on the environment.

In the DNS environment:

`hostname.domainname(.unique_number)` for Windows 95 `hostname(.unique_number)` for Windows 98 / NT

In the Non-DNS environment:

`hostname(.unique_number)` for Windows 95 / 98 / NT

These results are caused by the socket implementation. You can check the naming convention using a program that calls the `gethostname()` function as follows:

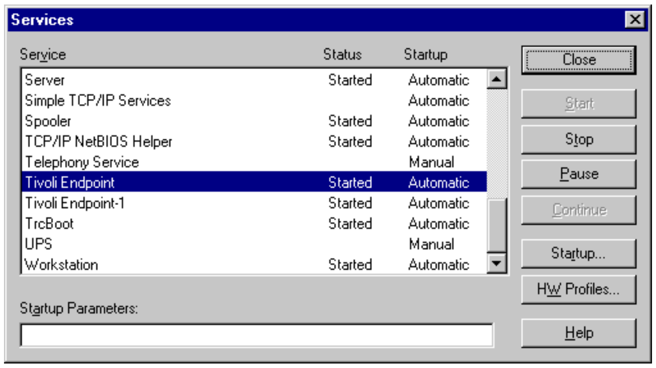
```
// hostname.cpp
//   Compile with
//   cl hostname.cpp wsock32.lib
#include <iostream.h>
#include <winsock.h>
void main()
{
    WORD wVersionRequested = MAKEWORD( 2, 2 );
    WSADATA wsaData;
    if (WSAStartup( wVersionRequested, &wsaData ) == 0) {
        char buf[255];
        gethostname(buf,sizeof(buf));
        cout << buf << endl;
        WSACleanup( );
    }
}
```

You can specify an arbitrary name to the Endpoint by passing an option of the `lcfd` command, `lcs.machine_name=<endpoint_label>`, when you install the Endpoint.

A unique number will be added to the end of the Endpoint label name when the same label name already exists. This sometimes can be caused by an Endpoint duplicate login.

3.7.1.2 Removing Endpoint Software

When you remove the Endpoint software from the Windows NT machine, you should check as to whether the Endpoint software has been removed completely or not. If you do not remove the Endpoint software completely and then reinstall the Endpoint on the machine, the Endpoint installation process creates the following NT services on the Windows NT:



Service	Status	Startup
Server	Started	Automatic
Simple TCP/IP Services		Automatic
Spooler	Started	Automatic
TCP/IP NetBIOS Helper	Started	Automatic
Telephony Service		Manual
Tivoli Endpoint	Started	Automatic
Tivoli Endpoint-1	Started	Automatic
TrcBoot	Started	Automatic
UPS		Manual
Workstation	Started	Automatic

Figure 36. The Tivoli Endpoint-1 NT Service

As you can see, the new NT service named Tivoli Endpoint-1 is running even if the former NT service (Tivoli Endpoint) is running on the same machine as well. In this case, the Endpoint will normally function normally and the unused service can be stopped; however, we recommend that you reinstall the Endpoint as follows:

1. Stop both Endpoint services.
2. Remove the Tivoli Endpoint-1 service using the following command:

```
C:\Tivoli\lcf\bin\w32-ix86\mrt\lcfd.exe -r lcfd-1
```

3. Remove the Endpoint software completely.
4. Reinstall the Endpoint software.

3.7.1.3 Broadcast to Different Subnet

Normally, the broadcast occurs within the same sub-network. You can force a directed broadcast by using `lcfd -g` option that accepts a colon-delimited list of IP addresses and ports. Any of these addresses can be a directed broadcast address. This also enables you to prioritize Gateways for initial logins.

3.7.1.4 Changing Endpoint Gateway Address

If your environment requires changing the IP address of the Endpoint gateway, use the following procedure:

1. Change the IP address of the Endpoint gateway and reboot the machine.
2. Run the following command:

```
wep set gateway -g <gw_name>
```

This sets the new IP address for all Endpoints which are currently logging into the Endpoint gateway. The Endpoints that are not logged in will follow the isolation scenario as described in the “Configuring the TMA Environment” on page 105 and the *Tivoli Framework Planning and Installation Guide*.

3.7.1.5 Login Request in Multiple Endpoint Gateways

If there are multiple Endpoint gateways on the same subnet, and the Endpoint is allowed to broadcast for the initial login, several of the Endpoint gateways may receive the request and process it. The result is that one Endpoint would log into multiple Endpoint gateways. The following shows the situation we mentioned:

```
/>wep ls
G      1189622596.4.21  grizzly-gateway
   1189622596.323.508+#TMF_Endpoint::Endpoint# salmon.323
G      1189622596.2.19  kodiak-gateway
   1189622596.224.508+#TMF_Endpoint::Endpoint# bass
   1189622596.245.508+#TMF_Endpoint::Endpoint# yi2250d
   1189622596.324.508+#TMF_Endpoint::Endpoint# salmon.324
G    1189622596.195.21  yi2250d-gateway
   1189622596.322.508+#TMF_Endpoint::Endpoint# salmon
/>wep salmon status
unable to determine endpoint status; endpoint may be unreachable.
/>
```

The last login request is honored, and the rest will return HMAC errors on downcalls and upcalls as follows. (PR-26857)

```
Dec 08 13:06:36 Q lcfd ** Exception caught in listener() (CNTL_EP): decrypt_data:
HMAC does not match encrypted data!
```

To avoid this problem, do not rely on broadcast as a means for the Endpoint’s initial login. Always use the select_gateway_policy or specify the Endpoint gateway during the Endpoint installation.

3.7.1.6 Binary Policies on Endpoint Gateways

Binary policies do not execute properly on Windows NT Endpoint Managers and Gateways. (PR-42284) To resolve this problem, you can use the policy script and invoke the binary in the script.

3.7.1.7 Shutting Down Using the lcfd.sh Command

Shutting down the Endpoint with the `lcfd.sh` command can produce the following error. (PR-44063)

```
Unable to get process id of the Tivoli LCF daemon (lcfd.pid).
```

3.7.2 Environment Variables and Files Considerations

In this section, we introduce the environment variables and files for each operating system.

3.7.2.1 Path Variable (Windows NT)

On a Windows NT system, using the `PATH` to define your system path makes NT commands unable to find the right environment and tools (`sh.exe`). The method scripts spawned by the oserv will hang. The variable should be set as `Path` (mixed-case) instead of `PATH` (uppercase).

3.7.2.2 windir Variable (Windows 3.x)

Some earlier versions of Windows do not have the `windir` variable set correctly. The InstallShield relies on this value. If it is missing, the InstallShield does not write the msvcrt40.dll file to the Windows directory and the `lcfd` daemon does not start correctly. Set the windir variable to the Windows directory and then run the InstallShield setup, or manually copy the msvcrt40.dll file to the Windows directory.

3.7.2.3 Installation Location (UNIX)

You cannot install the TMR Server, the Managed Node, or the Tivoli product into a location in the file system that has a path longer than 120 characters on a UNIX system. If you must use such a long path, set a symbolic link of the shorter length that points to the actual directory.

3.7.2.4 /tmp/.tivoli Directory (UNIX)

When you install Tivoli on UNIX, the directory called `/tmp/.tivoli` is created. This directory contains several files that are vital to the correct operation of the `oserv`. You should not erase this directory or any of its contents unless directed to do so by your Tivoli support provider. The installations should also ensure that regularly scheduled disk cleanup jobs (cron or Tivoli jobs) do not remove this directory.

3.7.2.5 Installation Location and lcf_env.bat Command

Some Tivoli applications download utilities to the Endpoint machines to be invoked from the command line by users working at those Endpoint machines. Before invoking such commands, the user must execute the batch file in the %SystemRoot%\Tivoli\lcf\1\lcf_env.bat to set up the required environment.

If the Endpoint is installed in a directory on the Windows 95 host machine that contains an embedded space, such as Program Files, the batch file in %SystemRoot%\Tivoli\lcf\1\lcf_env.bat does not set the `Path` environment variable correctly. When the set Path=%LCF_BINDIR%;%Path% statement is executed, the assignment fails with the following message: (PR-44284)

```
Too many parameters.
```

To correct this problem, edit the lcf_env.bat file to add quotation marks in the following line:

```
set Path="%LCF_BINDIR%";%Path%
```

To correct this problem remotely for multiple Windows 95 Endpoints, retrieve a copy of the file to edit with the following command:

```
wadminep <ep_label> view_file
C:\windows\Tivoli\lcf\%UNID%\lcf_env.bat > my_file.bat
```

Edit the file as stated above and replace the file with the following command:

```
wadminep <ep_label> send_file my_file.bat
C:\windows\Tivoli\lcf\%UNID%\lcf_env.bat
```

If all Windows 95 Endpoint have the same `$LCF_DATDIR`, replace the file for all of the Endpoints. Since the lcf_env.bat file contains references to the Endpoint installation directory path, different lcf_env.bat files must be used for Endpoints that use different installation directory paths.

3.7.2.6 Long Path Variable

The 16-bit programs (batch files) executed by the Endpoint or the application that is the client of the Endpoint can fail when the Path environment variable is too long. The Microsoft Windows 3.1 command interpreter limits the size of any environment variable string (VAR=VALUE) to 127 characters. With this limitation, the Path environment value cannot exceed 118 characters in length. (PR-43537)

The Endpoint automatically sets the Path for processes (`lcfd.exe`). By default, the Endpoint adds the following three directories to the Path when it starts:

- The path to the Endpoint library directory
- The path to the Endpoint cache directory
- The path to the Endpoint tools directory

Most Endpoint programs are 32-bit programs and run correctly with any setting of the Path. However, problems can arise when a 16-bit program is run and the Path value is greater than the limit supported by Microsoft Windows 3.1. Examples of Tivoli applications and services that invoke the 16-bit programs are:

- Tivoli Software Distribution (configuration program scripts)
- Tivoli Inventory
- Tivoli Task Library (batch files)

When the Endpoint executes a 16-bit program or batch file, the MS-DOS command interpreter (`COMMAND.COM`) is invoked to run the program or batch file. If the Path value that is passed to the `COMMAND.COM` is too long, the command interpreter will fail to execute correctly. The Endpoint will detect this situation and log the warning to the Endpoint log file (lcfd.log) if the Endpoint `log_threshold` level is set to level 1 (the default) or higher. The message logged is:

```
ERROR: The Path will be truncated by the command shell
```

To avoid this problem, install the Endpoint into the Tivoli recommended default location on the system (C:\Tivoli\lcf). Using this path means that the three values that are added to the Endpoint path at the start up will not cause the Path variable to exceed the 118 character limit.

Unfortunately, using a short installation path may not be enough. The Endpoint also passes the Windows 3.1 system environment (from AUTOEXEC.BAT) to the child processes. This can cause problems if the system Path on the user's machine has been modified to contain additional entries. This can happen when the user has installed other software packages on the system that have updated the Path in the AUTOEXEC.BAT.

When the path is too long, perform one of the following:

1. Reinstall the Endpoint to a shorter path on the system.
2. Edit the AUTOEXEC.BAT file. Add the following line:

   ```
   LASTDRIVE=Z
   ```

   Create the virtual drives for the long directory names in the Path:

   ```
   REM SET
   PATH=C:\DOS;C:WINDOWS;C:\SOME\OTHER\FOLDER
   REM change:
   SUBST S: C:\SOME\OTHER\FOLDER
   SET PATH=C:\DOS;C:\WINDOWS;S:\
   ```

   Reboot the machine for the changes to take effect.

3. Upgrade the Windows 3.1 system to Microsoft Windows 95 or Microsoft Windows NT. These versions of Windows do not suffer from the Path limitation problems when running 16-bit programs.

3.7.2.7 Upgrading Windows 95

Upgrading a Windows 95 Endpoint proceeds with a warning asking you to run the following command:

```
wadminep <ep_label> remove_endpoint 2
```

Running this command produces an error message. (PR-45158)

To resolve this problem, from the Start menu, open the **Run** dialog and run the `regedit` command. In the \HKEY_LOCAL_MACHINE| SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices directory you will find two entries: `lcfd` and `lcfd1`. Remove the `lcfd` entry.

3.7.2.8 Problems of the winstlcf Command

The `winstlcf -L "-n ep_label"` command may not work if a port other than default 9494 is used, or if the Endpoint is being reinstalled and the previous installation was not completely removed. In this case, the label becomes hostname+port. (PR-44451)

3.7.2.9 winstlcf Command Problem on Windows NT

Installing an Endpoint on a Windows NT host using the `winstlcf` command requires that the target host have the file share named `X$`, where `X` is the drive letter for the `%SystemRoot%`. The share must have full access granted to the user account specified in the `winstlcf` command. Furthermore, if the `-d` option is used to specify a non-default target installation directory, that directory must be on the same drive as `%SystemRoot%`. If the `X$` share is not on the target host, either create the share before using the `winstlcf` command, or use some other means, such as the InstallShield's `setup.exe`, to install the Endpoint. If you wish to install to a non-default target installation directory that is not on the same drive as `%SystemRoot%`, use a method other than the `winstlcf` command. (PR-43744)

3.7.2.10 winstlcf Command Problem on HP-UX and AIX

The `winstlcf` command fails when run in asynchronous mode (with the `-a` option) on the HP-UX and the AIX machines. The installations complete successfully and the Endpoints log into their specified Endpoint gateway, but the configuration directory (/etc/Tivoli/lcf/#) is not created. (PR-44378)

3.7.2.11 Problems regarding Batch Command on Windows 95

Batch tasks (*.bat) on the Windows 95 Endpoints do not return non-zero codes. Therefore, your script could fail and still return a zero code. (PR-44262)

To avoid this, write your .bat script to echo the failed message to standard output by using the `IF ERRORLEVEL` test condition after the commands.

3.7.2.12 Problems Regarding Batch Command on Windows NT

Tasks run from *.bat and *.cmd files on Windows NT 4.0 do not return information to the standard error or the standard output, even though the tasks execute successfully.

If you need this output, wrap the *.bat and *.cmd tasks in a sh.exe task, which returns standard error and standard output information. For example, the following script wraps the *.bat task:

```
#!/bin/sh
more > my.bat <<\MAIN_EOF
REM this is a comment
REM put your script ehre
REM for example a directory listing
dir
MAIN_EOF
cmd c/ my.bat
```

For shell scripts to run on Windows NT Endpoints, you must place the dependency on the sh.exe on the run_task method of the `TaskEndpoint` object. The dependency causes the sh.exe binary to be downloaded whenever the task is run on the Windows NT Endpoint, if it is not already there. To define the dependency, please refer to Chapter 8.6.7, “Using Dependencies to Deploy Tools to Endpoints” on page 318.

Chapter 4. Configuring the TMA Environment

In the classic Tivoli management environment, the installation process implicitly determined the TMR to which a managed system belonged and there was no requirement to explicitly configure the Managed Nodes to communicate with their TMR Server.

In the TMA environment, the Endpoint must be configured to communicate with an appropriate Endpoint gateway. To allow for both flexibility and availability, the association between the Endpoint and the Endpoint gateway can be dynamically determined and may change if a particular Endpoint gateway is not available.

In this chapter, we describe how you can control the TMA behavior and configure it in your TMA environment.

4.1 Overview of the TMA Login Process

When the Endpoint is started, it performs the Endpoint login process to participate in the TMR. There are two types of logins that the Endpoint performs: normal login and initial login.

4.1.1 Normal Login

When the Endpoint has already been a member of the TMR, the Endpoint performs the 'normal login upon startup. The normal login sequence is quite simple. The Endpoint logs into its assigned gateway and the Endpoint gateway acknowledges it. During the normal login, communication takes place only between the Endpoint and Endpoint gateway. The Endpoint sends subsequent login packets and communicates directly to the Endpoint gateway listed in the login information (lcf.dat) file. Since the Endpoint gateway has the Endpoint's information in its Endpoint list, communications are established immediately without contacting the TMR Server or the Endpoint Manager.

4.1.2 Initial Login

As a precondition for TMA operation, the Endpoint must become a member of the TMR. This process establishes the Endpoint as an active member of its TMR by assigning it to the Endpoint gateway. When the Endpoint service begins to run, it attempts to establish its region identity from the lcf.dat file. If this file does not exist, the Endpoint performs the initial login in order to establish that identity.

Three Step Process:
Initial login has the following three major phases:

- The Endpoint establishes communication with the TMR.
- The Endpoint Manager selects the Endpoint gateway for the Endpoint.
- The Endpoint receives its gateway assignment information and performs the normal login to the assigned gateway.

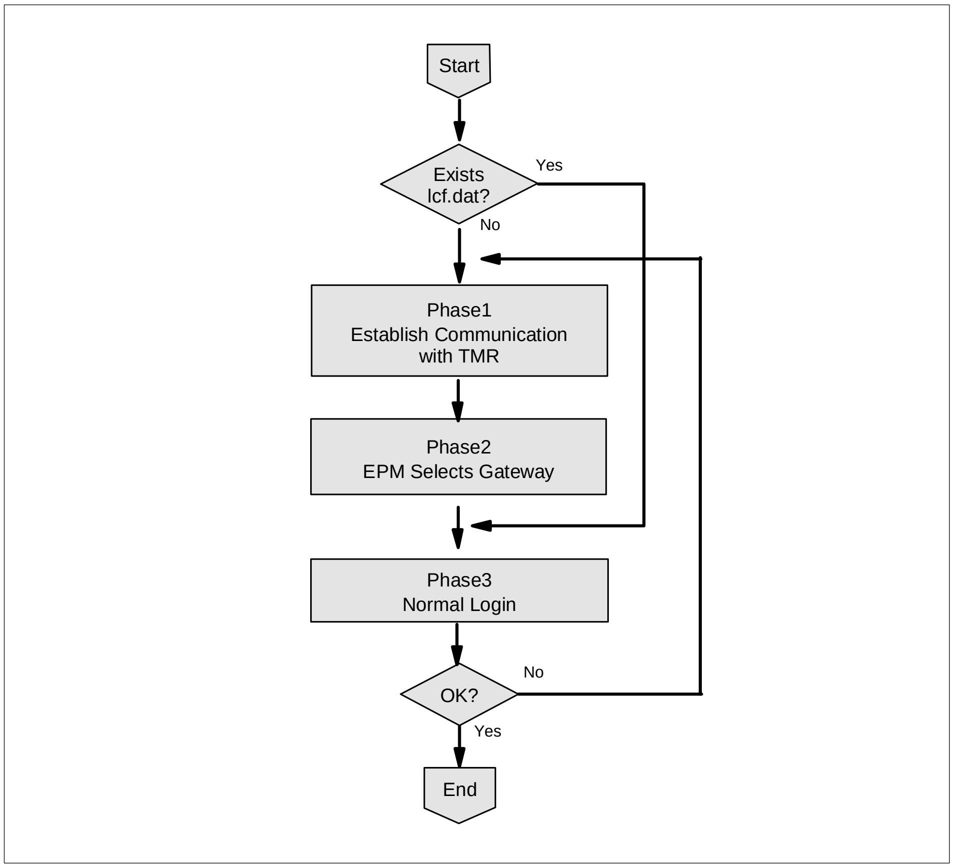


Figure 37. Initial Login Process

4.1.2.1 Finding a Region

The first step in the initial login process is finding an Endpoint gateway in the correct TMR. Finding a region means finding an Endpoint gateway to serve the initial login request. To do this, the Endpoint uses a set of network addresses configured during the installation process. This set of addresses is called the login interfaces or the gateway space of the Endpoint.

The Endpoint tries each address in turn until communication is successful. For each address, the Endpoint sends a UDP login packet and waits for the Endpoint gateway to respond by establishing a TCP connection for reply delivery. If a response is not received, the Endpoint will retry. The number of times it will retry and the amount of time it will wait between retries are user configurable through the udp_interval and udp_attempts parameters of the `lcfd` daemon. If the Endpoint receives no response after so many attempts, it moves to the next address in the login interfaces. The default values are five minutes for the udp_interval and six times for the udp_attempts.

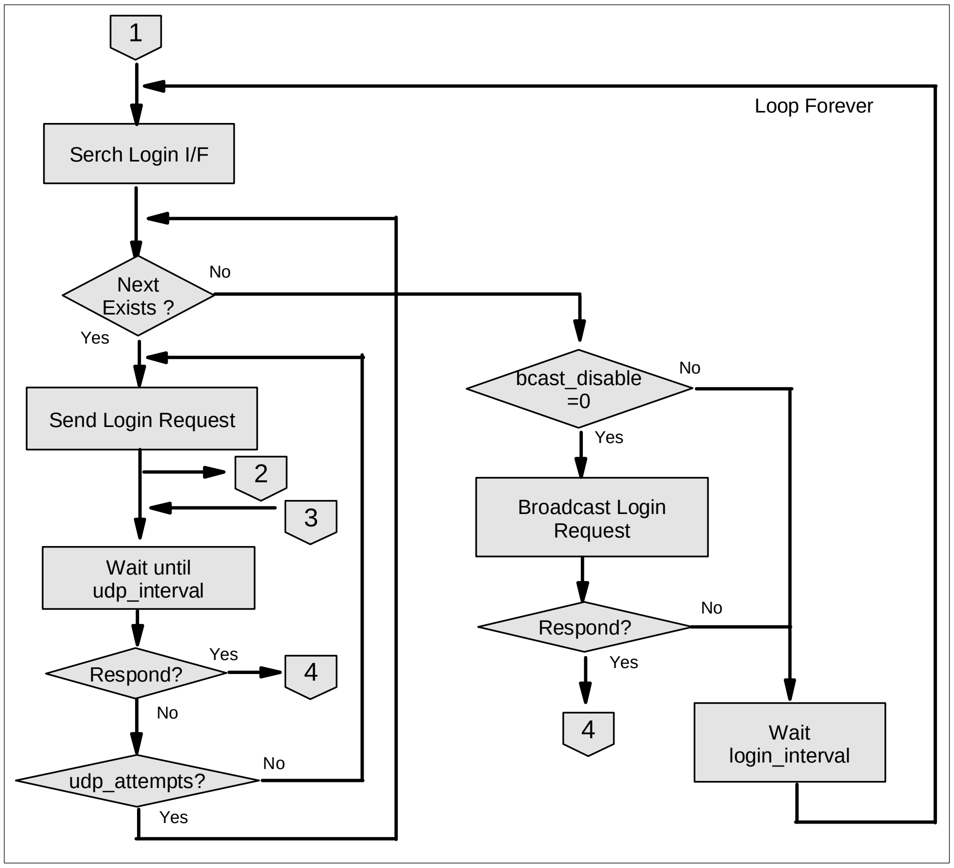


Figure 38. Finding a Region

4.1.2.2 Broadcast

If the entire login interface list is exhausted without establishing TMR communications, the Endpoint will send its initial login packet to the broadcast address. You can also place the broadcast address in the login interfaces list with the `-g` option of the `lcfd` daemon as follows:

```
-g 9.3.1.255
```

However, this operation is not recommended because the current version of the Tivoli Management Framework has a restriction related to Endpoint broadcasting described below. The Endpoint broadcast feature can be disabled by the bcast_disable=1 parameter of the `lcfd` daemon.

4.1.2.3 Rescue Failure Endpoint

What happens if the Endpoint fails to log in, even after trying every Endpoint gateway in it's login_interfaces and resorting to broadcast. In this case, multiple situations can be considered. In the following sections we introduce them.

login_interval

If the Endpoint fails to login, even after trying every Endpoint gateway in it's login_interfaces and resorting to broadcast, it pauses for some amount of time and then tries again. The duration of the pause is governed by the `lcfd` daemon parameter login_interval expressed in seconds. By default, this is set to 1800 (30 minutes). So if the Endpoint failed to log in because its Endpoint gateway didn't yet exist, or because of a transient routing problem, there is no problem. It will try again 30 minutes later. Because of this design, you can deploy the Endpoints at your leisure and bring up the Endpoint gateways, weeks, months or even years later when you are ready to begin managing them with the Tivoli Management Environment.

wep Command Options

While the Endpoint is pausing for the login_interval, it is listening for input. During this time, you can tell it what Endpoint gateway to use or even install a completely new set of login_interfaces. The `wep` command supports set gateway and set interfaces syntax for these purposes. As an extreme example, assume that your Endpoints have broadcast disabled and that every Endpoint gateway in their login_interfaces list suddenly crashed. You can recover from this by setting up new Endpoint gateways and using the `wep` command to first migrate Endpoints to the new Endpoint gateways and then to fix up their notion of assigned gateway and login_interfaces. The set gateway and set interfaces forms support the `-g` switch, which tells the Endpoint gateway to perform these operation for all of its Endpoints.

However, this is available only for an Endpoint that has performed the initial login successfully because you have to specify the label of the Endpoint with the `wep set gateway` and the `wep set interfaces` commands, and the Endpoint has to complete the initial login to get the Endpoint label. So, how can we modify the assigned gateway or the login_interfaces information for the Endpoint which has never performed the initial login? We will talk about this in the next chapter, Chapter 5, “Anatomy of TMA Behavior” on page 135.

4.1.2.4 Missing Features

The administrators and implementors should be aware that Version 3.6 of the Tivoli Management Framework does not distinguish between the multiple login requests that can arise from a single Endpoint using broadcast to locate the region. If multiple Endpoint gateways hear that broadcast request, then multiple requests will be forwarded to the Endpoint Manager. Since the Endpoint Manager does not recognize that these are from the same Endpoint, two Endpoint records will be created in the region. The Endpoint will still be managed, but it will exist under two labels and its assigned gateway with respect to the user-expected label is not predictable in advance. Only one of them (the last login request) is honored and the rest will return HMAC errors on downcalls and upcalls. If the two Endpoint gateways are in different TMRs, then the region is similarly undetermined. It is important to remember that specifying login interfaces and disabling broadcasting are the primary mechanisms for directing the Endpoint to the appropriate TMR during the initial login.

However, since most administrators do not enable UDP forwarding at their routers and each subnet may only have one Endpoint gateways in many environments, this is thought to be an acceptable risk given the benefits of a configuration-free environment. Nevertheless, a certain amount of study is a prerequisite for successful deployment of TMA across the enterprise.

4.1.2.5 Gateway Selection

When the Endpoint gateway receives an initial login request, it is said to be the Intercepting gateway for that request. The Intercepting gateway forwards the login request to the Endpoint Manager that is responsible for determining the best available Endpoint gateway for that Endpoint. The selection process is user-configurable, and we will describe it in detail later.

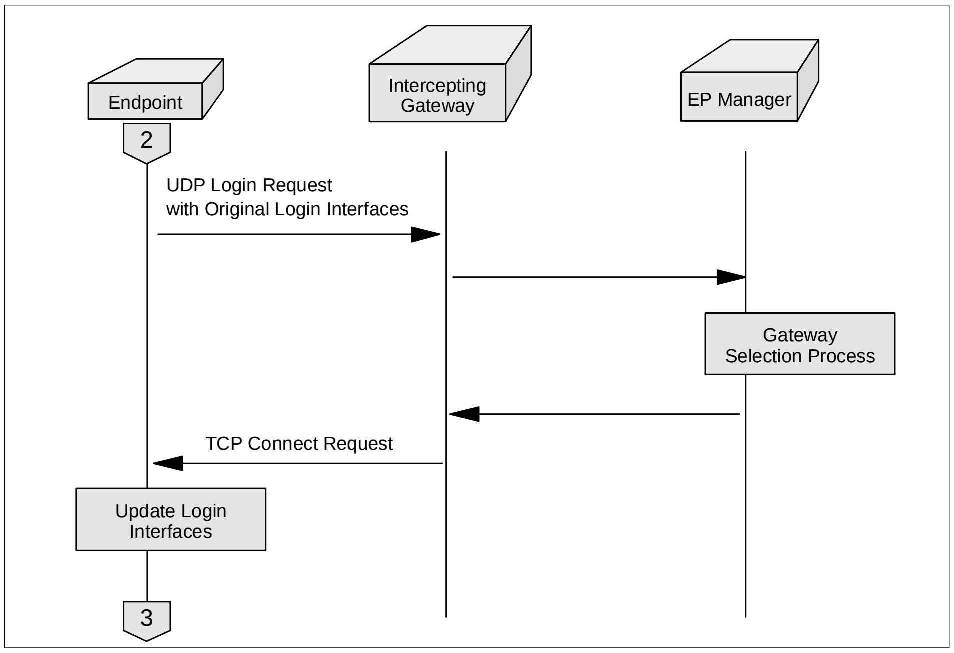


Figure 39. Gateway Selection (Sequence Chart)

4.1.2.6 Selected Login Interfaces

As a result of the Endpoint gateway selection process, several candidates for Endpoint gateways must be identified. These candidates may be different from the login interfaces given to the Endpoint at install time. This is possible because the login interfaces are part of the login request that flows between the Endpoint and the Intercepting gateway. If the Endpoint is later unable to contact its assigned gateway, it will fall back on the login interfaces list to reestablish connection with the TMR. This is said to be an isolation login. The Endpoint Manager assigns the Endpoint gateway, then adds the Endpoint information and gateway assignment into its Endpoint list. The Endpoint information is sent to the assigned gateway. The Intercepting gateway relays the assignment information to the Endpoint. As a result, the Endpoint performs the normal login to its assigned gateway.

4.1.2.7 First Normal Login

After receiving its gateway assignment from the intercepting gateway, the Endpoint performs the normal login to its assigned gateway. At this time, the login policy is run for the first time and the Endpoint codeset is downloaded to the Endpoint. The Endpoint is now ready to participate in Tivoli management operations.

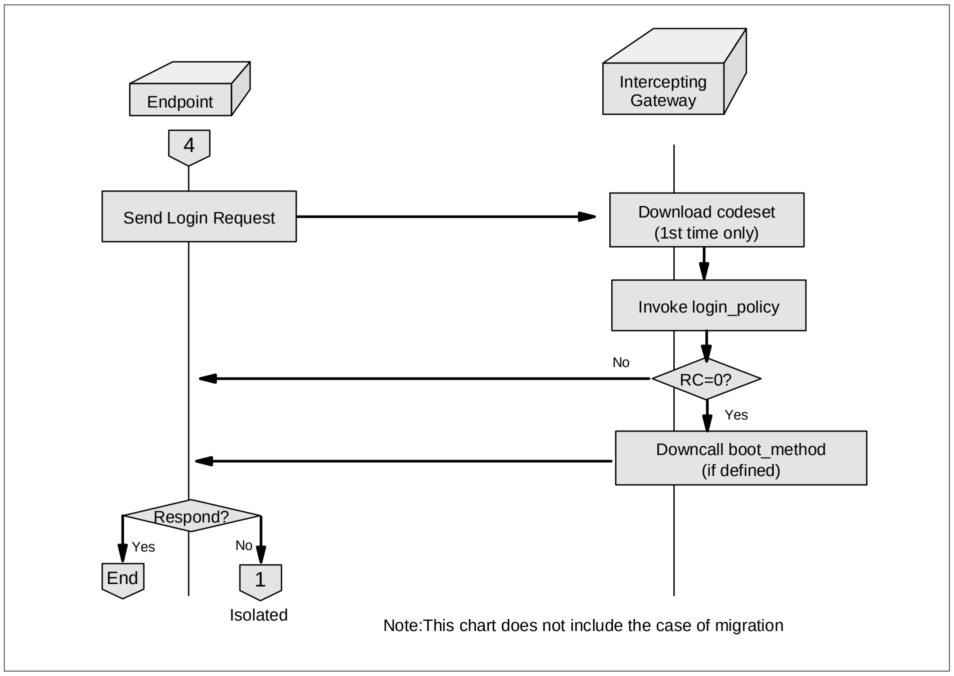

Endpoint
Intercepting Gateway
4
Send Login Request
Download codeset (1st time only)
Invoke login_policy
No
RC=0?
Yes
Downcall boot_method (if defined)
Respond?
Yes
No
End
1
Isolated
Note:This chart does not include the case of migration

Figure 40. Normal Login

You can configure boot_methods during the installation of the Endpoint or during the distribution of application profiles. The boot_method will run every time the Endpoint logs in to the Endpoint gateway. The boot_method can be developed using Tivoli ADE. Please refer to Figure 6 on page 211 for more information about the boot method.

4.1.3 Region Redirect

The Region redirect feature is a special case of the initial login. During the gateway selection process, it is possible for the user to specify the Endpoint gateway in another connected TMR. If it does, the gateway assignment is returned with a special status to tell the Endpoint to start the initial login process using the specified Endpoint gateway as the new intercepting gateway. By maintaining a well-known redirector TMR, you can gain some flexibility in Endpoint installations.

4.1.4 Isolation

When the Endpoint attempts to perform the normal login to its assigned gateway, for example, it may happen that the assigned gateway is unreachable. In this case, the Endpoint is said to be isolated. The Endpoint falls back to the initial login process of connecting to the TMR via the login interfaces determined at initial login. This is similar to the initial login except that for isolation, the Endpoint already exists. Therefore, the new Endpoint identity is not established. The Endpoint Manager simply performs the Endpoint gateway selection and returns the gateway assignment to the Endpoint via the intercepting gateway. Please refer to “TMR Redirection” on page 131 for more information about the TMR redirection.

4.2 Overview of Endpoint Policies and Configuration Files

You can configure the Endpoint's login behavior and communication patterns by developing scripts that execute at various times in the process. There are four such hooks. Of these, the login_policy runs on the Endpoint gateway; the other three run at the Endpoint Manager. These are the allow_install_policy, select_gateway_policy and after_install_policy. We will describe these Endpoint policies in the order in which they execute.

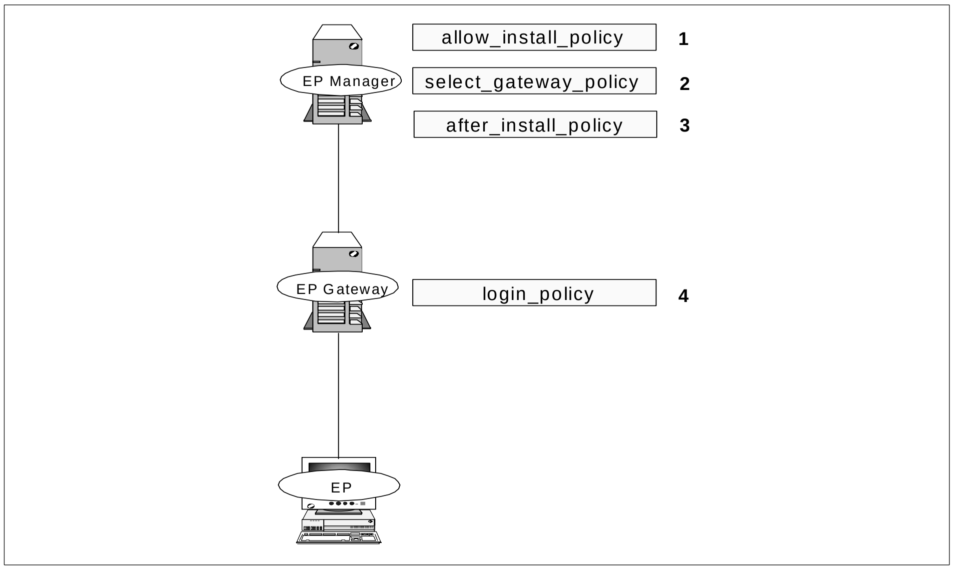


Figure 41. The Endpoint Policies

4.2.1 allow_install_policy

The allow_install_policy, or simply install policy for short, allows you to terminate the login immediately when the Endpoint Manager receives the login request from the intercepting gateway. For example, you can decide to refuse the login request based on the Endpoint's IP address for example. To do this, simply exit the script with a non-zero value. You can also use this policy to perform any pre-login actions you might need.

Note

For post 3.6 releases of the Tivoli Management Framework, special meaning is assigned to the exit code of 6 from the allow_install_policy. In this case, the Endpoint Manager will instruct the Endpoint gateway to ignore the login request. This has been used in the field to help manage some broadcast scenarios.

4.2.2 select_gateway_policy

The select_gateway_policy (SGP for short) is unique in that it is the only Endpoint policy script that is specified to produce output. The output is a list of Endpoint gateway object references that is used by the Endpoint Manager to make the gateway assignment for that Endpoint. The Endpoint Manager will assign the Endpoint to the first available Endpoint gateway. This same list is also returned to the Endpoint for use later if the assigned gateway is unavailable. In this later context, the Endpoint gateway list is called the ’login_interfaces or the gateway space.

The Endpoint policy overrides the Endpoint Manager's default selection process and it is recommended to use Endpoint policy in multiple gateway environments. The Endpoint Manager tries to contact each Endpoint gateway in the order listed in the Endpoint policy script. The first Endpoint gateway that the Endpoint Manager contacts successfully is the Endpoint gateway to which the Endpoint is assigned. The intercepting gateway is also added to the end of the login interfaces list to ensure that the Endpoint has at least one definite contact. If the Endpoint gateways listed in the script cannot be contacted, the Endpoint Manager assigns the intercepting gateway to the Endpoint.

Note

The variable of `LCF_LOGIN_STATUS` is also set by the Endpoint Manager. The value of 2 indicates the Endpoint is isolated. It means that the Endpoint was unable to contact its assigned gateway. The isolated Endpoint is automatically migrated to another Endpoint gateway unless the select_gateway_policy terminates with a non-zero exit status.

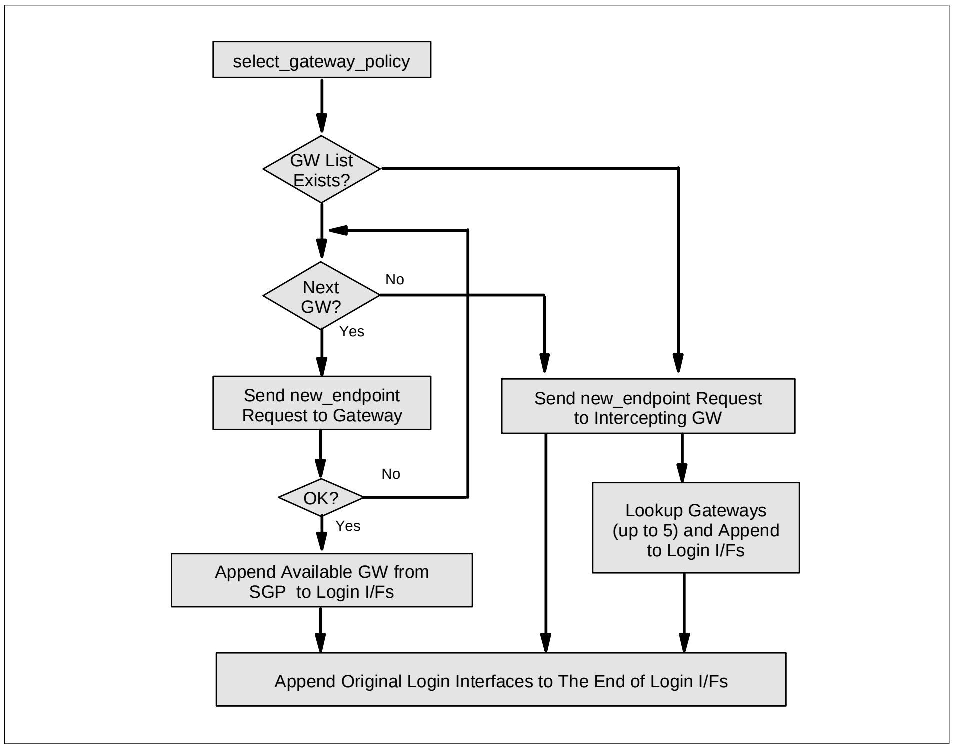

select_gateway_policy
GW List Exists?
Next GW?
No
Yes
Send new_endpoint Request to Gateway
OK?
No
Yes
Append Available GW from SGP to Login I/Fs
Send new_endpoint Request to Intercepting GW
Lookup Gateways (up to 5) and Append to Login I/Fs
Append Original Login Interfaces to The End of Login I/Fs

Figure 42. Gateway Selection and New Login Interfaces

In the above figure, we tried to clarify how the new login interfaces would be created after running the Endpoint gateway selection process on the Endpoint Manager. The following figure (Figure 43 on page 114) shows how the Endpoint Manager creates the Endpoint gateway list.

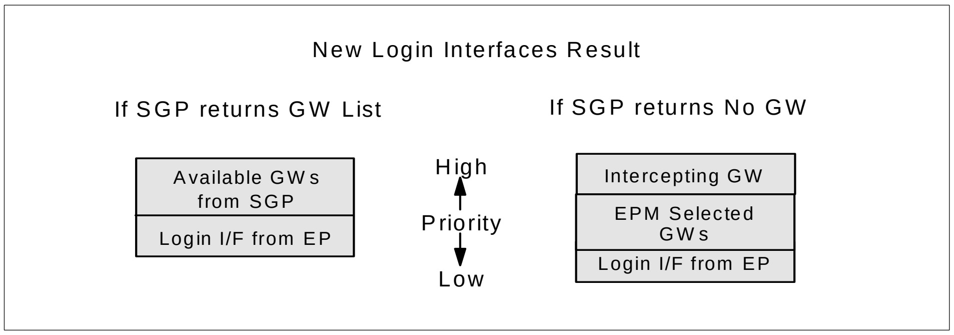


Figure 43. New Login Interfaces from Endpoint Manager

4.2.3 after_install_policy

The after_install_policy script is invoked after the gateway assignment is made but before the Endpoint is notified. Management operations such as a downcall cannot be invoked on the Endpoint machine at this time. The standard example for the after_install_policy is to add the Endpoint to a policy region. This Endpoint policy is run after the initial login only; it will not be run on subsequent Endpoint logins.

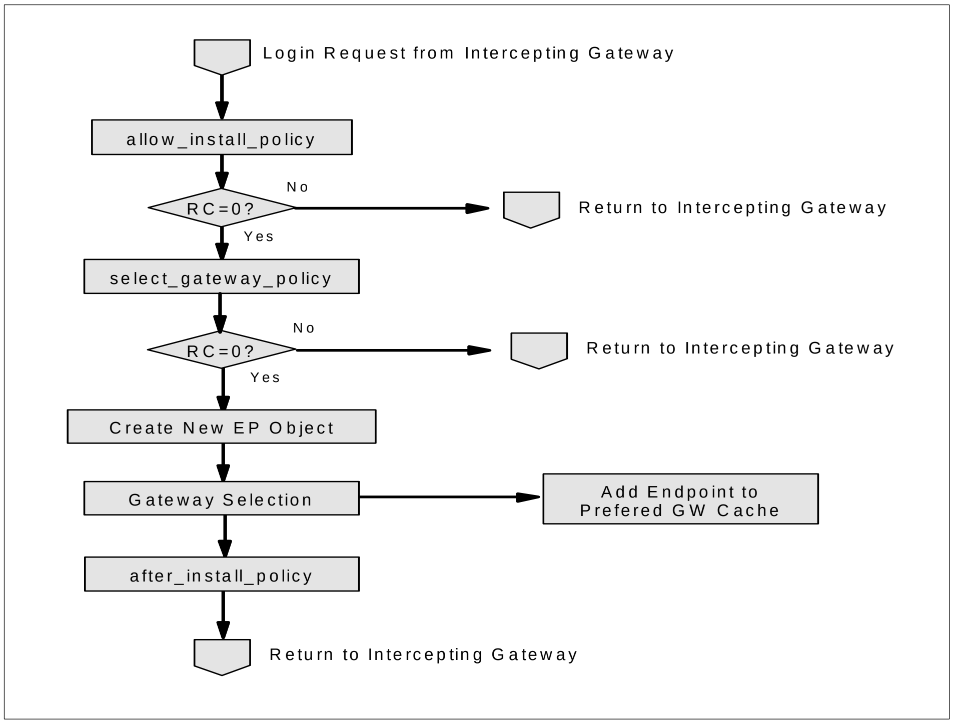

Login Request from Intercepting Gateway
allow_install_policy
No
RC=0?
Yes
Return to Intercepting Gateway
select_gateway_policy
No
RC=0?
Yes
Return to Intercepting Gateway
Create New EP Object
Gateway Selection
Add Endpoint to Prefered GW Cache
after_install_policy
Return to Intercepting Gateway

Figure 44. Policies Running on Endpoint Manager

4.2.4 login_policy

This policy is run on the Endpoint gateway upon every normal login and performs any action you need. For example, the login_policy can be useful when you configure the auto upgrade function of the Endpoint software. If the login_policy exits with a non-zero value, the Endpoint login will not fail, but the Endpoint gateway doesn’t invoke the boot_method (refer to “TMA and Tivoli Management Applications” on page 211 for more information about the boot method).

Note

The same login_policy script is run on all of the Endpoint gateways in the TMR. This policy doesn't support the use of binaries.

4.2.5 Policy Arguments

The scripts that are defined in the Endpoint policy all have the same argument conventions and ship with a default script, exit 0, which means that no special policy applies. This policy and its functional equivalents are known as trivial policy. The scripts have the same argument conventions, as illustrated by the following excerpt from the default Endpoint policy.

```
#
# The following are the command line arguments passed to this script
# from the Endpoint Manager.
#
# $1 - The label of the Endpoint machine
# $2 - The object reference of the Endpoint machine
# $3 - The architecture type of the Endpoint machine
# $4 - The object reference of the  gateway that the Endpoint logged into
# $5 - The ip address of the Endpoint logging in.
# $6 - region
# $7 - dispatcher
# $8 - version
```

These arguments don't all exist at every point in the process. For example, the Endpoint IP address ($5) is always defined. However, the object reference of the Endpoint machine ($2) is not created until after the allow_install_policy exits with a value of 0. So $2 has the value of `OBJECT_NIL` when passed to the allow_install_policy script.

We can run the following script to see all of these arguments in the Windows and AIX operating system environments.

```
#!/bin/sh
LOGFILE=/tmp/policylog
printline() {
    echo  ‘date +"%Y/%m/%d %H:%M:%S: ‘ $* >> $LOGFILE
    return
}
printline "[allow_install_policy]"
printline "The label of the ep machine: $1"
printline "The object reference of the ep machine: $2"
printline "The architecture type of the ep machine: $3"
printline "The object reference of the gateway: $4"
printline "The ip address of the ep logging in.: $5"
printline "region: $6"
printline "dispatcher: $7"
printline "version: $8"
printline "LCF_LOGIN_STATUS=${LCF_LOGIN_STATUS}"
#sleep 2
printline "Exitting ..."

exit 0
```

We will introduce the result of this script for each platform. Please refer to “Endpoint Policy Argument Values” on page 395 for more information.

4.2.6 Policy Exit Code

In this section, we introduce what information would be logged when the Endpoint policy exits with a non-zero return code. In this example, we set the debug level of the Endpoint gateway to 9.

4.2.6.1 Exit with Non-Zero from allow_install_policy

In the case of an exit with a non-zero code from allow_install_policy, the following messages appear in the epmgrlog file:

```
1998/12/08 10:44:55 +06: exception servicing login:
The allow_install policy script did not execute properly or exited with a non-zero return code.
```

The following messages appear in the gatelog file:

```
1998/12/08 10:45:04 +06: failure during login for 9.3.1.193+1066
(salmon,w32-ix86,LFVCN74D2LF7L3YQCPTQ00000584,reg=0,od=0):
The allow_install policy script did not execute properly or exited with a non-zero return code.
```

D
R
A
F
T

4.2.6.2 Exit with Non-Zero from select_gateway_policy

In the case of an exit with a non-zero code from select_gateway_policy, the following messages appear in the epmgrlog file:

```
1998/12/08 10:42:25 +06: gateway selection denied with status: 1
1998/12/08 10:42:25 +06: exception servicing login: epmgr select policy had nonzero status
```

The following messages appear in the gatelog file:

```
1998/12/08 10:42:34 +06: failure during login for 9.3.1.193+1061
(salmon,w32-ix86,XBH2HN4M7RZK9MRR06G700000567,reg=0,od=0): epmgr select policy had nonzero status
```

4.2.6.3 Exit with Non-Zero from login_policy

In the case of an exit with a non-zero code from login_policy, the following messages appear in the gatelog file.

```
1998/12/08 10:51:50 +02: run_login_policy: Running login policy on endpoint salmon.
1998/12/08 10:51:53 +02: failure during login for 9.3.1.193+1072
(salmon,w32-ix86,58MQVSK1LCN20XDXWYB100000586,reg=1189622596,od=309):
12/08/98 10:51:53 (26): command exited with signal 1, core=FALSE
```

For your reference, the following messages appear in the gatelog file when the login_policy exits with a 0. As you can see, in this case, `run_ep_boot_methods` was executed (refer to “TMA and Tivoli Management Applications” on page 211 for more information about the boot method).

```
1998/11/23 15:12:16 +06: run_login_policy: Running login policy on endpoint salmon.
1998/11/23 15:12:19 +06: run_ep_boot_methods: nothing to do.
```

4.2.7 Applying Policies

The CLIs for manipulating the Endpoint policies are the `wputeppol` and `wgeteppol` commands. The Tivoli Management Framework is installed with default Endpoint policy scripts. They don’t contain any logic, and simply return the value of zero to the Endpoint Manager or Endpoint gateway. To implement useful Endpoint policies for your environment, you need to replace the default Endpoint policies with your own scripts. The following three basic steps are required:

- Retrieve the current Endpoint policy script using the `wgeteppol` command.
- Modify this script to add the appropriate logic.

- Replace the current Endpoint policy script as the new script using the `wputeppol` command.

The `wputeppol` and `wgeteppol` commands take a single parameter specified by the Endpoint policy whose script you are retrieving or replacing. The script itself is routed to stdout for the `wgeteppol` command and read from the stdin for the `wputeppol` command. Therefore, you would normally use redirection as shown in the example below. For more information regarding the `wgeteppol` and `wputeppol` commands, refer to the *Tivoli Framework Reference Manual*.

When you want to customize the allow_install_policy, invoke the following command:

```
wgeteppol allow_install_policy > allow_install_policy.sh
```

This command will create the file, called allow_install_policy.sh, including the current policy script. Then this file can be edited to include the logic such as the samples shown in this section.

After saving the modified shell script, execute the following command to enable the policy for subsequent Endpoint login processes:

```
wputeppol allow_install_policy < allow_install_policy.sh
```

All policy scripts are stored in the Tivoli object databases, so that it is possible to keep consistency between the Endpoint Manager and Endpoint gateways. In this way, all Endpoint gateways receive the latest login_policy script. Note that the same login_policy script would be applied to all Endpoint gateways in the TMR. If the logic defined in the login_policy depends on a specific operating system, the login_policy script must have the logic that determines on which platform it is executing and act accordingly.

Note

You don't have to pass the fully Endpoint policy name to the `wputeppol` and `wgeteppol` commands. Instead, you can pass unique abbreviations. For example, to retrieve the allow_install_policy you can say:

```
wgeteppol al
```

4.2.8 Configuration Files

Endpoints would use two files to store configuration information. These files are located under the dat subdirectory of the Endpoint installation directory (typically C:\Tivoli\lcf\dat\1).

4.2.8.1 lcf.dat File

The lcf.dat file contains the login information related to the Endpoint. Since this file is a binary file, you cannot edit this file directly. However, when you start the `lcfd` daemon you can overwrite certain information in this file using the `lcfd` command arguments. Once the Endpoint connects to its assigned gateway, the Endpoint gateway address, port number, any network aliases for the assigned gateway and alternate gateway information is stored in the lcf.dat file.

4.2.8.2 last.cfg File

All other configuration information is stored in the last.cfg file. Once the Endpoint and Endpoint gateway are connected, for example the initial login completed, the configuration information is stored in the last.cfg file. For every subsequent startup, the startup command for the `lcfd` daemon (`lcfd` or `lcfd.sh`) retrieves the configuration information from the lcf.dat and last.cfg file. The following is an example of the last.cfg file:

```
lcfd_port=1029
lcfd_preferred_port=9494
gateway_port=9494
log_threshold=2
start_timeout=120
run_timeout=120
lcfd_version=5
logfile=C:\Tivoli\lcf\dat\1\lcfd.log
config_path=C:\Tivoli\lcf\dat\1\last.cfg
run_dir=C:\Tivoli\lcf\dat\1
load_dir=C:\Tivoli\lcf\bin\w32-ix86\mrt
lib_dir=C:\Tivoli\lcf\bin\w32-ix86\mrt
cache_loc=C:\Tivoli\lcf\dat\1\cache
cache_index=C:\Tivoli\lcf\dat\1\cache\Index.v5
cache_limit=20480000
log_queue_size=1024
log_size=1024000
udp_interval=30
udp_attempts=2
login_interval=120
lcs.crypt_mode=196608
```

The last.cfg file contains the most recent configuration information. You can alter this file to affect the options that will be used the next time the Endpoint is started. For more information, refer to the manual, the *Tivoli Framework Reference Manual*.

To modify the configuration of the Endpoint, you can either, edit the last.cfg file or restart the Endpoint using the startup command (`lcfd` or `lcfd.sh`) with the appropriate arguments. If you start the Endpoint from the command line interface using the `lcfd` or `lcfd.sh` command, the arguments you specify override the equivalent entries in the last.cfg file. If you edit the last.cfg file, the new configuration information would be used when you restart the Endpoint (`lcfd` daemon). Once the Endpoint performs the Endpoint login, the configuration information is stored in the last.cfg file again. You can also use the Endpoint web interface to modify the contents of the last.cfg file.

4.3 Customizing EP Policies

In this section, we introduce some useful examples of the Endpoint policies for your reference.

4.3.1 Example of allow_install_policy

The following is an example of the allow_install_policy script. This example doesn’t allow the Endpoints on the subnet 9.3.2 to log in to the TMR. It also doesn’t allow the Endpoints that don’t have a host name to perform the Endpoint login.

```
#!/bin/sh
set -e
#
# Don’t allow endpoints from subnet 9.3.2 log into this TMR.
#
SUBNET=‘echo $5 | awk -F"." ’{ print $1"."$2"."$3 }’‘
if [ "$SUBNET" = "9.3.2" ]; then
    exit 1
fi
#
# Don’t allow endpoints who does not have a name
#
if [ "$1" = "" ]; then
    exit 1;
fi

exit 0
```

The following example allows only the Endpoints found in the file `eplist.txt` to perform the Endpoint login:

```
#!/bin/sh
#
# Allow only the endpoints found in eplist.txt
. /etc/Tivoli/setup_env.sh
while read x
do
if [ $1 = $x ]; then
exit 0
fi
done < $DBDIR/eplist.txt
exit 1
```

4.3.2 Example of select_gateway_policy

The following example returns all Endpoint gateways that are on the same subnet as the Endpoint:

```
#!/bin/sh
#
# only ep_ip is needed for this example
ep_ip=$5
#
FOUNDONE=FALSE
# we just want the subnet of the endpoint
SUBNET=‘echo $ep_ip | cut -d’.’ -f3‘
# get all gateways and find ones that are on the same subnet
GATEWAYS=‘wlookup -ar Gateway -o‘
for gwoid in $GATEWAYS
do
    gwproxy=‘idlattr -tg $gwoid proxy Object‘
    mnips=‘wifconfig -h $gwproxy | grep -v Device | awk ’{print $2}’‘
# a managed node might have multiple interfaces, so check
# each of them if the gateway subnet matches the endpoint
# subnet, return gwoid if it matches
    for ip in $mnips
    do
        wsub=‘echo $ip | cut -d’.’ -f3‘
        if [ $gwsub -eq $SUBNET ]; then
        # echo $gwsub such as ’1189622596.4.21#TMF_Gateway::Gateway#’
            echo $gwoid
            FOUNDONE=TRUE
        fi
    done
done
# if you did not find a gateway, and you still want the
# endpoint to log in, exit 0, else exit 1
if [ "$FOUNDONE" = "TRUE" ]; then
    exit 0
else
    exit 1
fi
```

4.3.3 Example of after_install_policy

The following example subscribes a new Endpoint to a profile manager that represents the Endpoints which are of similar architecture type. If the policy region or profile manager doesn’t exist, this after_install_policy creates them.

```
#!/bin/sh
#
LCF_POLICY_REGION=LCF-Endpoints
PROFILE_MANAGER=LCF-$3
EP=$1
#
# Check to see if our top-level policy region already
# exists.If not create it and put it on this administrators
# desktop.
#
# Disable "exit on error" for this call since we will handle
# the failure.
#
set +e
wlookup -r PolicyRegion $LCF_POLICY_REGION > /dev/null
ERR=$?
set -e
if [ $ERR -ne 0 ]; then
    ALI=‘objcall 0.0.0 get_security_objid‘
    set ‘objcall $ALI get_identity‘
    ADMIN="$1"
    ADMIN_OID="$2"
    wcrtpr -m ProfileManager -a $ADMIN $LCF_POLICY_REGION
    idlcall $ADMIN_OID refresh_collection
fi
```

Figure 45. Example of after_install_policy (part 1 of2)

```
#
# Check to see if our interp specific profile manager
# already exists. If not create it and make it dataless so
# that we can subscribe the endpoint to it.
#
# Disable "exit on error" for this call since we will handle
# the failure.
#
set +e
wlookup -r ProfileManager $PROFILE_MANAGER > /dev/null
ERR=$?
set -e
if [ $ERR -ne 0 ]; then
    wcrtprfmgr $LCF_POLICY_REGION $PROFILE_MANAGER
    wsetpm -d /Library/ProfileManager/$PROFILE_MANAGER
fi
#
# Subscribe the endpoint to the profile manager which
# contains the endpoints for that specific interp type.
#
wsub /Library/ProfileManager/$PROFILE_MANAGER @Endpoint:$EP

exit 0
```

Figure 46. Example of after_install_policy (part 2 of 2)

After running this policy, the Endpoint would be subscribed to a profile manager. The following figure (Figure 47) shows an example of a Windows NT Endpoint (`w32-ix86`).

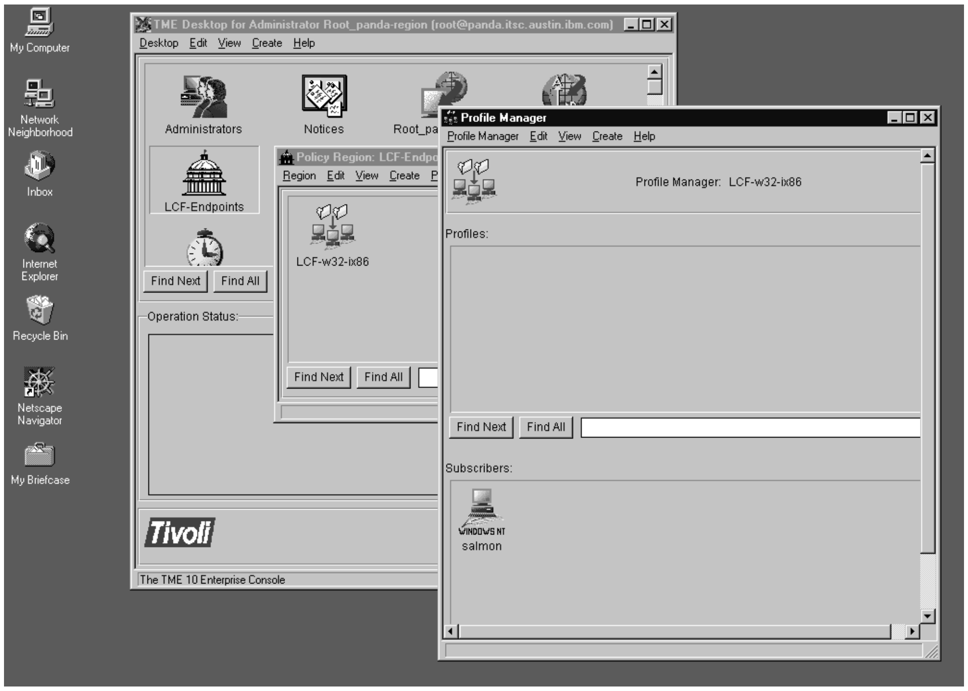


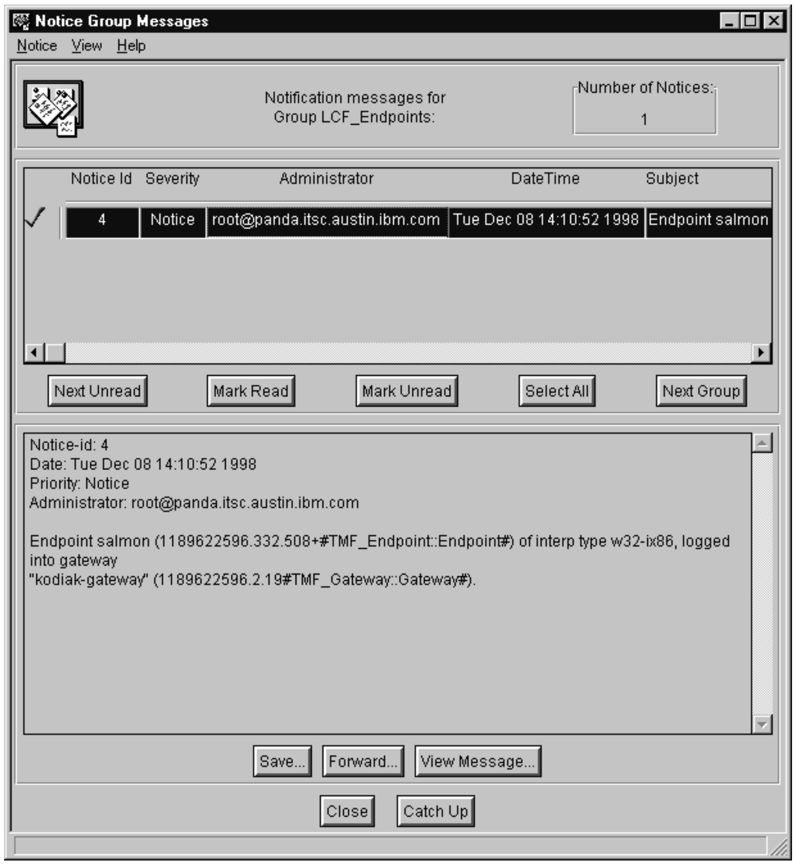
Figure 47. Endpoint Subscription Using Policy

4.3.4 Example of login_policy

The following example sends a Tivoli notice to the LCF-Endpoints notice group every time the Endpoint logs in to the Endpoint gateway and automatically upgrades the Endpoint software (`lcfd`).

```
#!/bin/sh
#
# Invoke the upgrade script to check the current version of
# the endpoint software and upgrade if necessary.
BO=‘objcall 0.0.0 self‘
OS=‘objcall $BO getattr oserv‘
INSTALLDIR=‘objcall $OS query install_dir|tr ’\\\\’ ’\’‘
$INSTALLDIR/lcf_bundle/upgrade/upgrade.sh $1 $8 $3
#
LCF_NOTICE_GROUP=LCF_Endpoints
#
# Send a notice to LCF endpoint notice group every time this
# endpoint logs in.
#
set +e
wlookup -r TMF_Notice $LCF_NOTICE_GROUP > /dev/null
ERR=$?
set -e
if [ $ERR -ne 0 ]; then
    NTFGM=‘wlookup -r Classes TMF_Notice‘
    idlcall -T top $NTFGM \
        TMF_Notice::NoticeManager::create_notice_group \
        ’"’$LCF_NOTICE_GROUP’" 72’
fi
GW=‘idlcall $4 _get_label‘
EPOID=‘wlookup -o -r Endpoint $1‘
wsndnotif $LCF_NOTICE_GROUP Notice << LCF_NOTICE
Endpoint $1 ($EPOID) of interp type $3, logged into gateway
$GW ($4).
LCF_NOTICE
exit 0
```

After running this policy, the Tivoli notice would appear in the appropriate notice group as follows:



Notice Id	Severity	Administrator	DateTime	Subject
4	Notice	root@panda.itsc.austin.ibm.com	Tue Dec 08 14:10:52 1998	Endpoint salmon

```
Notice-id: 4
Date: Tue Dec 08 14:10:52 1998
Priority: Notice
Administrator: root@panda.itsc.austin.ibm.com

Endpoint salmon (1189622596.332.508+#TMF_Endpoint::Endpoint#) of interp type w32-ix86, logged
into gateway
"kodiak-gateway" (1189622596.2.19#TMF_Gateway::Gateway#).
```

Figure 48. Log to Notice Using Policy

The following example simply logs the Endpoint login information to the log file, eplogin.log:

```
#!/bin/sh
#
. /etc/Tivoli/setup_env.sh
LOGFILE=$DBDIR/eplogin.log

println() {
    echo  ‘date +"%Y/%m/%d %H:%M:%S: ‘ $* >> $LOGFILE
    return
}

println "Endpoint $1 ($5) logs in"
exit 0
```

4.4 Gateway Migration

You can change the gateway assignments made at initial login. When the Endpoint changes its gateway assignment from Endpoint gateway A to Endpoint gateway B, it is said to have migrated from A to B. Use the `wep` command to reassign the assigned gateway as follows:

```
wep ep_label migrate gw_label
```

where

`ep_label`	Specifies the Endpoint to be migrated
`gw_label`	Specifies the Endpoint gateway to which the Endpoint is reassigned

Refer to the *Tivoli Framework Reference Manual* for more information about the `wep` command.

The command to migrate the assigned gateway prompts the Endpoint Manager to update the Endpoint list on both the new assigned gateway and the formerly assigned gateway. For scalability reasons, and because the Endpoint may not be reachable at the desired migration time, the Endpoint is not contacted by default at the time of the migration. Hence, the Endpoint gateways must expect to receive the normal login requests from the Endpoint that they no longer manage. When this happens, the formerly assigned gateway acts as an intercepting gateway during the initial login -- obtaining the new gateway assignment from the Endpoint Manager and relaying this to the Endpoint, which then performs the normal login to the new Endpoint gateway. The Endpoint is said to be ‘migrating’ until it learns of its new gateway assignment. The actions undertaken by the Endpoint during this time are called ‘migratory’. When the Endpoint learns of its new gateway assignment, the migration is said to be complete.

The following figure (Figure 49 on page 129) shows the process flow of the Endpoint gateway migration.

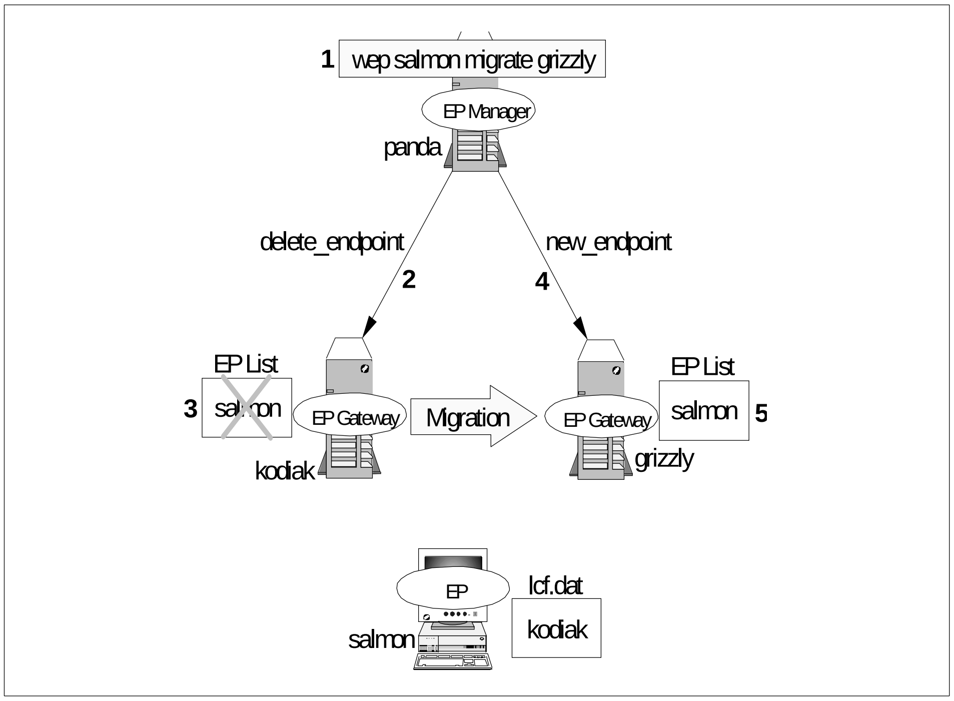


Figure 49. Gateway Migration

1. Execute the `wep migrate` command.
2. The `delete_endpoint` method is issued.
3. Delete the Endpoint from the original Endpoint gateway's cache.
4. The `new_endpoint` method issued.
5. Add the new Endpoint to the new Endpoint gateway's cache.

As you can see, during the Endpoint migration process, there is no communication with the target Endpoint itself.

4.4.1 Migration Completion

The Endpoint can be told of its new gateway assignment at the time of migration. By default, the Endpoint doesn't immediately discover its new gateway assignment. The Endpoint can make the discovery in one of the following ways:

- Performing a management operation (upcall)
- Performing a management operation (downcall)

- Endpoint login (such as when the Endpoint is rebooted)

For example, the `wep set gateway -e ep label` command is a good way to update the Endpoint after migration. But, it is basically a downcall. Therefore, it is called a migration completion by downcall.

4.4.1.1 Completion by Migratory Login

The handling of the login request from the migrated Endpoint to its formerly assigned gateway is similar to the case of the completion by migratory upcall. The former gateway intercepts the Endpoint's login request, then obtains the new gateway assignment for the Endpoint from the Endpoint Manager. This information is forwarded to the Endpoint. The Endpoint then performs the normal login to its new assigned gateway.

4.4.1.2 Completion by Migratory Upcall

An ’upcall is when the management application running on the Endpoint invokes a method on the Managed Node hosting its Endpoint gateway. For security reasons, the method request passes through the Endpoint service that forwards it to the Endpoint gateway. When the Endpoint is migrating, it sends the request to the formerly assigned gateway which is considered the intercepting gateway for the upcall because the Endpoint gateway no longer manages the Endpoint. The intercepting gateway determines the current gateway assignment for the Endpoint and returns this to the Endpoint. The Endpoint performs the normal login and resends the upcall request to the new assigned gateway. The Endpoint gateway selection process does not occur at the Endpoint Manager because the migration will normally have occurred recently.

4.4.1.3 Completion by Migratory Downcall

A downcall is when the management application running on the Endpoint gateway host invokes a method on the Endpoint. This means the Endpoint gateway is able to automatically receive the migration information from the Endpoint Manager, so that a downcall issued by the newly assigned gateway would reach the Endpoint without a problem. For each downcall, the Endpoint checks the peer address of the requesting Endpoint gateway. If the address has changed, the Endpoint makes a note of this and rewrites its lcf.dat file. The subsequent Endpoint logins and upcalls will proceed normally. If the address change resulted from a migration, the Endpoint is able to avoid the failover logic described before in the migratory upcall and login section.

Note

This process is not supported when the NAT device separates gateways and Endpoints. The NAT environments must rely on upcall and login to complete the migration process.

4.4.1.4 Completion by Isolation Login

If the Endpoint attempts to contact its previous Endpoint gateway prior to discovering its new gateway assignment, and if the previous Endpoint gateway is unreachable, then the migration will complete by the isolation login. At that time, the Endpoint is said to be isolated, so that the Endpoint would use its login information list (lcf.dat file) and attempt to perform the Endpoint login to an alternate gateway. If the Endpoint fails to log in to all of the alternate gateways, the Endpoint attempts to perform the isolated login using a broadcast packet.

Note

The select_gateway_policy is run for isolation logins. Hence, if the migration is performed via the `wep` command, and if that migration is inconsistent with the select_gateway_policy, then the gateway assignment will change if the migration completes by the isolation login. Of course, nothing will happen if there is no select_gateway_policy.

4.5 TMR Redirection

The TMR redirection is one of the solutions for managing multiple TMR environments, and it makes the Endpoint configuration easy. In this part of book, we introduce what the TMR redirection is and its advantages.

4.5.1 Redirectors

Version 3.6 of the Tivoli Management Framework supports the feature known as TMR redirection. This feature allows you to set up a special TMR to function as the master router for the Endpoint logins across many regions.

4.5.2 How It Works

The normal select_gateway_policy script outputs Endpoint gateway object references as candidate gateways for the Endpoint logging in. When you use TMR redirection, nothing changes. Simply output Endpoint gateway object references for Endpoint gateways in any region of your choosing as follows:

```
echo ‘wlookup -r Gateway trout-gateway‘
```

or

```
echo ‘wlookup -r Gateway trout-gateway#panda-region‘
```

The Endpoint Manager understands that these references are to Endpoint gateways in the remote region and will behave accordingly. The only requirement is that the redirector gateway must be interconnected to all regions referenced in the policy.

Upon the completion of the select_gateway_policy, when the Endpoint Manager detects that remote object references have been specified, it simply rewrites the Endpoint login_interfaces and sets a special flag to tell the Endpoint about the redirection. This data is relayed back to the Endpoint, which then behaves as if it had been restarted with the `-g x` option, where `x` is the login_interfaces specified by the redirecting select_gateway_policy.

Note that Endpoint gateway selection did not actually occur. The Endpoint simply gets a new set of gateways to use as interceptors in the (second round) initial login process. Once the Endpoint contacts the new region, the initial login process proceeds as described above. In particular, any select_gateway_policy in the new region is honored.

If this policy is trivial, the default behavior, as described above, is to make the intercepting gateway the assigned gateway. In this way, one can manage all gateway selections from a single policy script. Moreover, one can create the hierarchy of redirectors to benefit from delegation. You can also create redirection cycles, so be careful.

4.5.3 Simplified Endpoint Configuration

TMR redirection can simplify Endpoint configuration in environments where the broadcast approach is not adequate because one does not wish to deploy an Endpoint gateway in every broadcast space.

If the pure broadcast approach won’t work, some form of Endpoint configuration is necessary, most likely in the form of using the login_interfaces. This induces coupling between the original Endpoint machine configuration (perhaps at the factory) and the planning of the Endpoint gateway gateways. It may not be possible or desirable to know the names or addresses of the Endpoint gateways when the Endpoint machine is initially configured.

To address this problem, you can choose to configure masses of Endpoints to look for a well-known name, such as tivoli-gateway. That is, the Endpoint might be setup such that the Endpoint comes up using the `-g tivoli-gateway` command line. The machine called `tivoli-gateway` could represent a single machine that is reachable from all Endpoint networks. Or it could represent a family of machines that serve a similar role in multiple DNS domains. Such a machine would be a good candidate for the Endpoint gateway in the redirecting TMR.

4.5.4 Sharing Endpoint Resources across TMRs

The Endpoint, Endpoint gateway and Endpoint Manager resources can be exchanged across connected TMRs. These resources can be shared to enable distributing file packages from all Tivoli applications and running any tasks supported by the Endpoint. Direct management of the Endpoint (such as migrating Endpoints to new Endpoint gateways or listing the Endpoints assigned to an Endpoint gateway) must, however, be performed in the Endpoint's local TMR. Management commands such as, `wgateway`, `wep`, and `wdelep` must be run locally.

If you share the Endpoint resource across multiple TMRs, you must also share the Endpoint Manager resource. The Endpoint gateway resource can also be shared, but there is little benefit in doing so. We, therefore, recommend that you do not share the gateway resource.

Note

As we mentioned, TMR redirection or sharing Endpoint resources across TMRs is an efficient solution in multiple TMR environments. However, you should consider the TMR design before you install masses of Endpoints because a single TMR is able to handle several thousands of Endpoints in Version 3.6 of Tivoli. The important thing here is that the TMR redirection feature is available only for the multiple TMR environment. We will introduce TMR design in Chapter 9, "Management Examples Using TMA" on page 359.

4.6 Conclusion

The TMA login process is complex. Its features and failure modes must be studied carefully before attempting any sort of rollout. In the next chapter, we introduce the detailed behavior of TMA. After reading both chapters, you will understand TMA behavior well. For example, behaviors such as the login process, isolation, and migration should become clear.

Chapter 5. Anatomy of TMA Behavior

A detailed understanding of the Endpoint login process and general Endpoint behavior can be very helpful, especially for problem determination The previous chapter introduced you to configuring and controlling the Endpoint login, however, there can be additional considerations that complicate this process. In this chapter, we provide detailed information regarding the Endpoint login and Endpoint behavior.

5.1 Our Test Environment

To verify the behavior of the TMA, we performed many tests during the project that resulted in this redbook. First, we introduced our test environment. The following figure (Figure 50) shows our environment for performing the tests we will discuss.

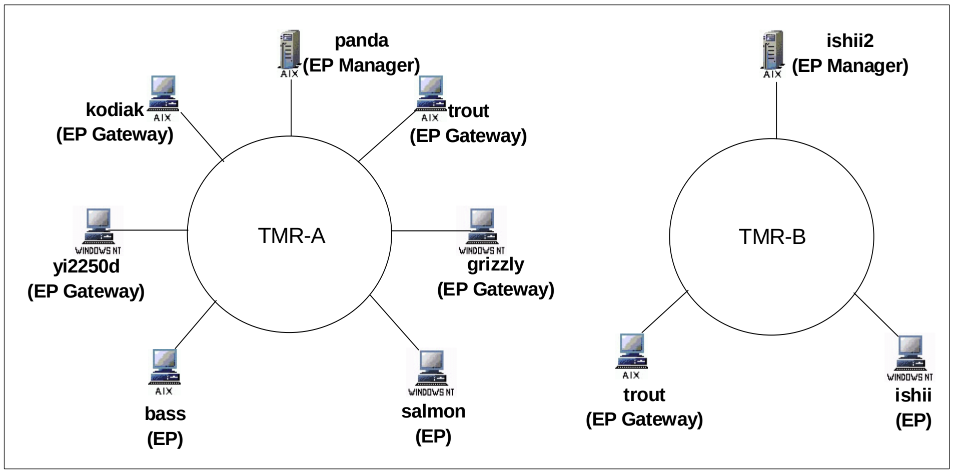


Figure 50. ITSO Austin Test Environment for the TMA Project

Note

We changed the configuration of the host trout a few times, so that trout appears in both TMRs: TMR-A and TMR-B (depending on the test case). However, trout is actually just one AIX machine.

The following table shows an overview of the machines shown in the Figure 50 on page 135.

Table 11. The Overview of the Test Machines

Host Name	Management Resource Type	Operating System
panda	TMR Server (Endpoint Manager)	AIX V4
ishii2	TMR Server (Endpoint Manager)	AIX V4
kodiak	Managed Node (Endpoint gateway)	AIX V4
trout	Managed Node (Endpoint gateway)	AIX V4
grizzly	Managed Node (Endpoint gateway)	Windows NT V4
yi2250d	Managed Node (Endpoint gateway)	Windows NT V4
bass	Endpoint	AIX V4
ishii	Endpoint	Windows NT V4
salmon	Endpoint	Windows NT V4
		Windows 98
		Windows 95

We performed tests covering many scenarios using the above test machines during our project.

5.1.1 Our Test Scenario

To understand and verify TMA behavior, we assumed many situations in our test environment. The following table (Table 12 on page 136) shows our testing cases. We will discuss the results and points of interest for each case throughout this chapter.

Table 12. The Test Scenario in the TMA Project

Case	Category	Test
Case 1	-g Option	The -g option function test.
Case 2	-g Option	The -g option function test.
Case 3	wep Command	Modify the assigned gateway.
Case 4	wep Command	Modify the login interfaces.
Case 5	Broadcast	The broadcast function test.
Case 6	Broadcast	The broadcast function test.

Case	Category	Test
Case 7	Broadcast	The broadcast function test.
Case 8	Web Interface	Modify the assigned gateway.
Case 9	TMR Redirection	The TMR redirection function test.
Case 10	Initial Login	Assume the EP Manager is unavailable.
Case 11	Initial Login	Assume the EP gateway is unavailable.
Case 12	Initial Login	Assume the EP gateway is unavailable.
Case 13	Initial Login	The select_gateway_policy function test.
Case 14	Initial Login	The select_gateway_policy function test.
Case 15	Normal Login	Assume the EP Manager is unavailable.
Case 16	Normal Login	Assume the Endpoint isolation.
Case 17	Normal Login	Assume the Endpoint isolation.
Case 18	Normal Login	Assume the Endpoint isolation.
Case 19	After Login Completion	Issue the upcall.
Case 20	After Login Completion	Issue the downcall.
Case 21	After Login Completion	Issue the upcall.
Case 22	After Login Completion	Issue the downcall.
Case 23	Migration	The migration function test.
Case 24	Migration	The migration completion by the EP login.
Case 25	Migration	The migration completion by the upcall.
Case 26	Migration	The migration completion by the downcall.
Case 27	Migration	The migration completion by the isolate login.

For detailed information regarding each scenario, refer to the following sections.

5.2 Understanding Options to Control Endpoint Login

There are several ways to control Endpoint login, as we introduced before. We classified this information to make understanding it easier before we provide a detailed explanation. The following are options and directions for controlling the TMA.

lcfd -g	Specifies the Endpoint gateway in the login interface list when the `lcfd` starts.
lcfd -D lcs.login_interfaces	Specifies the Endpoint gateway in the login interface list when the `lcfd` starts.
wep set gateway	Modifies the assigned gateway.
wep set interfaces	Modifies the login interface list.
wep migrate	Modifies the Endpoint list located in Endpoint Manager and Endpoint gateway.
select_gateway_policy	The policy executed on the Endpoint Manager for deciding the assigned gateway and alternate gateway.

Basically, we combined the above options, then started the `lcfd` daemon with the options for testing. We will discuss the result of the tests we performed in the next few sections.

5.3 Tracing TMA Behavior

In the previous chapter, Chapter 4, “Configuring the TMA Environment” on page 105, we explained the Endpoint login, how to configure the TMA, how to control the TMA, and so on. In this section, we will show TMA behavior in a real environment. All examples introduced in this chapter are based on the result of tests we performed during our project.

5.3.1 Using -g Option for Endpoint Initial Login

The `lcfd` has many options for controlling its own behavior. The -g option is the typical option for controlling the Endpoint initial login, and specifies to which Endpoint gateway the Endpoint will attempt to perform an initial login. In this part of the book, we validate the behavior of the `lcfd` daemon with the -g option during the initial login. We tested the following two cases regarding the -g option of the `lcfd` daemon and received the following results:

Case 1:
This is a very basic test and the most typical usage for the -g option. To avoid broadcasting, we should always use the -g option as follows:

Table 13. Case 1: The -g Option Function Test

Case 1	
Scenario	Specify the EP gateway (kodiak) for initial login.
Option	`lcfd -g kodiak+9494`
EP Policy	None
Result	The EP logged into the EP gateway (kodiak) we specified.

Case 2:
In this case, we specified the broadcast address of the subnetwork that the Endpoint was connected to. As you can see from this example, we can specify the broadcast address using the -g option even if it is different from the subnetwork which the Endpoint belongs to.

Table 14. Case 2: The -g Option Function Test

Case 2	
Scenario	Specify the broadcast address to the same subnetwork.
Option	`lcfd -g 9.3.1.255`
EP Policy	None
Result	The EP attempted to broadcast to the same subnetwork for initial login.

Note

When you specify the broadcast address as in Case 2, the `lcfd` daemon cannot recognize it as the broadcast address. Therefore, the `lcfd` daemon attempts to broadcast for the Endpoint login even if you define the bcast_disabled=1 (disable) option. We don’t recommend that you specify the broadcast address with the -g option.

5.3.2 Using -D lcs.login_interfaces Option for Endpoint Initial Login

Since the -D lcs.login_interfaces option of the lcfd daemon is the same as the -g option, there is nothing new to introduce here. Version 3.2 of the Tivoli Management Framework provided only this option (-D lcs.login_interfaces) to specify the Endpoint login interfaces. The -g option has been added by Version 3.6 of the Tivoli Management Framework.

Note

When you specify the -D lcs.login_interfaces and -g option at the same time, what happens?

```
lcfd -D lcs.login_interfaces=kodiak+9494 -g grizzly+9494
```

The first argument (`-D lcs.login_interfaces`) is handled first, then the second argument (`-g`) is handled. Therefore, the following sample has the same meaning as the above sample.

```
lcfd -g kodiak+9494:grizzly+9494
```

or

```
lcfd -D lcs.login_interfaces=kodiak+9494:grizzly+9494
```

Normally, we recommend you use the -g option because it is more simple. However, there is no functional difference between the -D lcs.login_interfaces and the -g option.

5.3.3 Using Other Options for Controlling lcfd Daemon

There are many options for controlling the lcfd daemon. In this project, we tested the following options and checked the behavior of the lcfd daemon.

-D log_threshold	Defines the level (from 0 to 4) of debug messages written to the lcfd.log file. This option has the same function as the -d option.
-D udp_attempts	Specifies the number of times the Endpoint will transmit an initial login request.
-D udp_interval	Specifies the number of seconds between Endpoint initial login request attempts.
-D login_interval	Specifies the waiting period before the Endpoint executes another login attempt. The Endpoint login attempt will retry indefinitely using this interval value.

All options we tested worked correctly. These options can be defined in the last.cfg file, setup.iss file or using the Web interface.

5.3.4 Using the wep Command for Modifying Login Information

The `wep` command provides a feature to modify the assigned gateway information and the login interfaces list information that is contained in the lcf.dat file located in the Endpoints.

5.3.4.1 What is the lcf.dat File?

The lcf.dat file is created by the `lcfd` daemon when the Endpoint logs in to the Endpoint gateway successfully. This file contains two pieces of very important information regarding the Endpoint login. One is the assigned gateway information and the other is the login interfaces list information. The assigned gateway is the Endpoint gateway that the Endpoint has logged into and the login interfaces list is the list of the alternate gateways where the Endpoint will attempt to perform the Endpoint login when the assigned gateway is not available. The `wep` command modifies this information as shown in Figure 51.

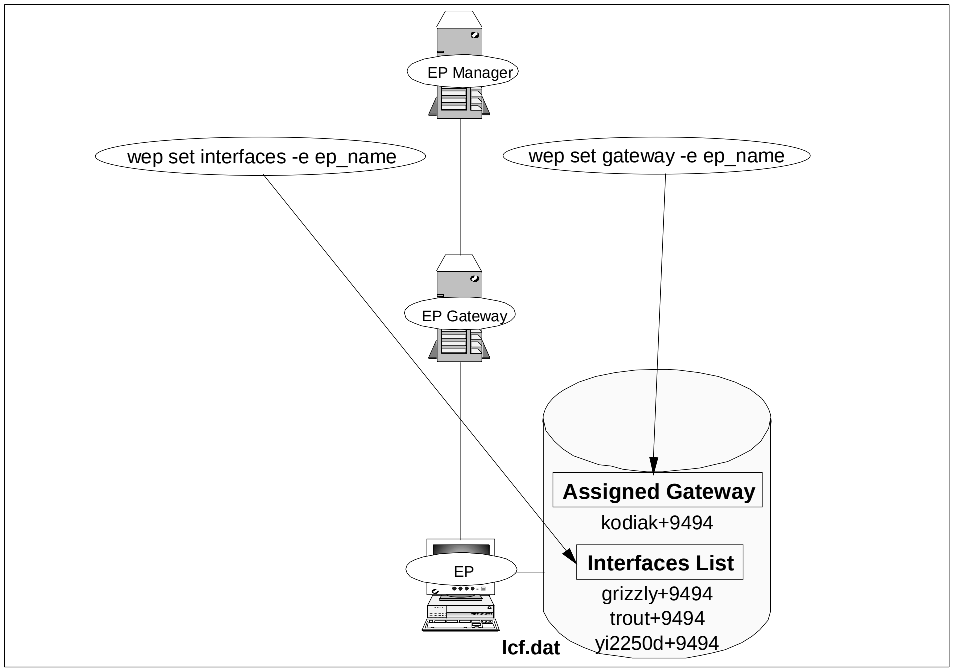
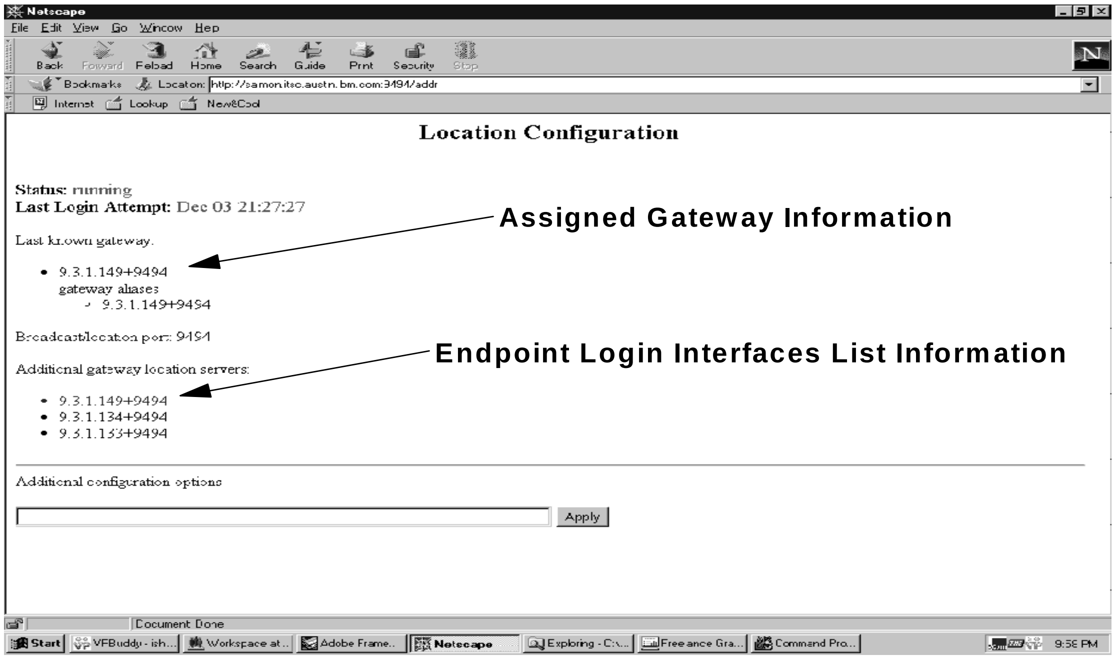


Figure 51. The wep Command Used to Modify Endpoint Login Information

5.3.4.2 Browsing Endpoint Login Information

How do we browse the Endpoint login information? We can simply use the Endpoint Web interface to browse the Endpoint login information. To browse the login information, choose the **Network Address Configuration** page from the LCF Daemon Status Page. On this page, the Last known gateway: field shows the assigned gateway and the Additional gateway location servers field shows the login interfaces list.


Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://salmon.itsc.austin.ibm.com:9494/addr
Internet Lookup NewCool
Assigned Gateway Information
Endpoint Login Interfaces List Information
Apply
Document Done
Start
9:56 PM

Location Configuration

Status: running
Last Login Attempt: Dec 03 21:27:27

Last known gateway:

- 9.3.1.149+9494
  gateway aliases
  - 9.3.1.149+9494

Broadcast/location port: 9494

Additional gateway location servers:

- 9.3.1.149+9494
- 9.3.1.134+9494
- 9.3.1.133+9494

Additional configuration options

Figure 52. Browsing the Endpoint Login Information

5.3.4.3 Testing the wep Command to Modify Login Information

We tested the following two cases regarding the `wep set gateway` and `wep set interfaces` commands to verify the behavior of the `lcfd` daemon.

Case 3:

We modified the assigned gateway information of the Endpoint (salmon) using the `wep set gateway` command, then made sure of the result by using the Endpoint Web interface.

Table 15. Case 3: The wep set gateway Command Test

Case 3	
Scenario	Modify the assigned gateway information using the wep command, then verify the result using the Endpoint web interface.
Command	`wep set gateway -e salmon`
EP Policy	None
Result	The assigned gateway information was modified correctly.

Case 4:
This is a simple test and is very similar to Case 3. In this case, we modified the Endpoint login interfaces list information of salmon, then verified the result.

Table 16. Case 2: The wep set interfaces Command Test

Case 4	
Scenario	Modify the login interfaces list information using the wep command, then verify the result using the Endpoint Web interface.
Command	`wep set interfaces -e salmon trout+9494`
EP Policy	None
Result	The login interfaces list information was modified correctly.

In this case, if you execute the `wep ls` command, the following output will appear.

```
# wep ls
G      1189622596.4.21  grizzly-gateway
   1189622596.323.508+#TMF_Endpoint::Endpoint# salmon.323
G      1189622596.2.19  kodiak-gateway
   1189622596.224.508+#TMF_Endpoint::Endpoint# bass
   1189622596.245.508+#TMF_Endpoint::Endpoint# yi2250d
   1189622596.324.508+#TMF_Endpoint::Endpoint# salmon.324
G    1189622596.195.21  yi2250d-gateway
   1189622596.322.508+#TMF_Endpoint::Endpoint# salmon
# wep salmon status
unable to determine endpoint status; endpoint may be unreachable.
```

As you can see, each Endpoint gateway processes the Endpoint login request. Since the last login request is honored, the other Endpoint entries are not available. Therefore, the `wep ep_label status` command returns the unreachable status.

5.3.5 Using Broadcast for Endpoint Login

As we have mentioned, broadcast should normally be disabled in the configuration of the TMA. The reason is that the broadcast from the TMA might affect the network that the TMA is connected to and make the performance worse. Although, it should be noted that the broadcast packets are small and sent relatively infrequently. In this project, we tested a few cases regarding broadcast for understanding the behavior of the TMA.

Case 5:
In this case, we configured three Endpoint gateways on the same subnetwork before testing. In other words, it was a multiple Endpoint

gateways environment. Then we started to install the TMA (`lcfd`) with default options. By default, the broadcast was enabled, so that the `lcfd` began to perform broadcast for initial login. Then, several of the Endpoint gateways received the login request from the Endpoint and processed it, so that one Endpoint logged in to multiple Endpoint gateways for a single initial login. Therefore, we can see one Endpoint in multiple Endpoint lists using the `wep ls` command. However, only the last login request is honored.

Table 17. Case 5: The Broadcasting Test in Multiple EP Gateways Environments

Case 5	
Scenario	Install the lcfd with default option for performing the initial login in the multiple EP gateways environment.
Option	Default (`-D bcast_disable=0`)
EP Policy	None
Result	The Endpoint performed the initial login, then several of the EP gateways received the login request and processed it. Therefore, one Endpoint was added to more than one EP list. However, only one EP list had been working correctly.

Note

This behavior is considered a software defect and will be fixed in the next patch. This is also noted in the *Framework Release Notes V3.6 September 1988*.

Case 6:

This case is almost the same as Case 5, but we used broadcasting in a multiple TMR environment. We configured two TMR servers (Endpoint Managers) on the same subnetwork and also configured two Endpoint gateways before testing. Then we started to install the `lcfd` with the default options and the `lcfd` began to perform the broadcast for the initial login. Then, Endpoint gateways that belong to different TMRs received the login request from the Endpoint and processed it, so that one Endpoint logged in to multiple Endpoint gateways for the single initial login. Therefore, we can see one Endpoint in multiple TMRs using the `wep ls` command. However, the last login request processed is honored by the Endpoint. So, even though the Endpoint may show up in multiple TMRs, it is only manageable by the last one. This is also considered a defect and will be fixed in the near future.

Table 18. Case 6: The Broadcasting Test in the Multiple TMR Environment

Case 6	
Scenario	Install the `lcfd` with default option for performing the initial login in the multiple TMR environment.
Option	Default (`-D bcast_disable=0`)
EP Policy	None
Result	The Endpoint performed the initial login, then several of the EP gateways that belong to different TMRs received the login request and processed it. Therefore, one Endpoint had been added to more than one EP list and appeared on the both TMRs on the same subnetwork. However, only one EP list had actually been working correctly.

Case 7:

In this case, we disabled the broadcast (`bcast_disable=1`) in the setup.iss file.Then we started to install the `lcfd` without the login interface list (-g option) configuration. Since there was no login interfaces list, the Endpoint tried to send login requests using the broadcast, but the broadcast was disabled, so the Endpoint couldn’t send any login requests. In this case, the Endpoint couldn’t log in to any Endpoint gateways.

Table 19. Case 7: The bcast_disable Option Function Test

Case 7	
Scenario	Define the bcast_disable=1 option (broadcast is disabled) and install the `lcfd` for performing the initial login.
Option	`-D bcast_disable=1`
EP Policy	None
Result	Nothing happens. The Endpoint couldn’t log in to any EP gateway.

Note

As you can see from the result of Case 7, you have to define the login interface information using the -g or -D lcs.login_inrerfaces option when you define the broadcast as disabled (`bcast_disable=1`).

When you meet this situation (Case 7), is there any way to resolve it? The answer is yes. Using the Endpoint Web interface is one solution for this case. You cannot use the `wep` command in this case because the Endpoint has not yet completed the initial login, so the Endpoint doesn't have the Endpoint label. We have to specify the Endpoint label for modifying the login interface information using the `wep` command.

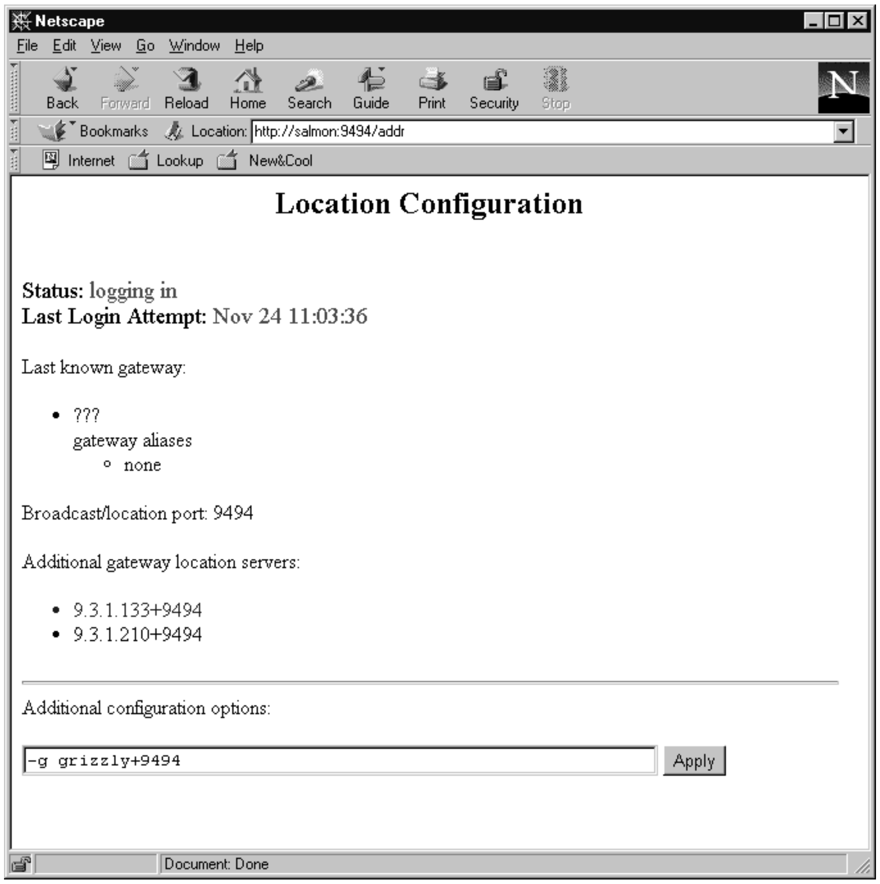
5.3.6 Using the Web Interface

As an alternative to the `wep` command, you can use the Web browser to reconfigure the TMA. The `lcfd` daemon remains resident, listening on the default port, even if the Endpoint initial login attempt fails. We can use the Web browser to access the daemon on the Endpoint and enable the Endpoint to log into the correct or new Endpoint gateway. To use the Endpoint Web interface, you will need to know the hostname, port, user ID and password for the Endpoint. These are all obtainable via the `wep` command. Use the URL `http://host:port` to connect as follows:

```
http://salmon.itsc.austin.ibm.com:9494
```

Click on **Network Address Configuration**. Type in `options` in the box, just as you would on the `lcfd` command line. Click **apply**.

Case 8:
The following figure shows a sample of the Location Configuration page in the Endpoint Web interface. In this figure, the status field shows logging in status and the Last known gateway field shows ???. This means that the Endpoint is attempting to perform the initial login, but has not completed it yet. When the Endpoint gateways, which are specified as the login_interfaces, are not reachable and the broadcast function of the Endpoint is configured as disabled, this situation can happen.


Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://salmon:9494/addr
Internet Lookup New&Cool

Location Configuration

Status: logging in
Last Login Attempt: Nov 24 11:03:36

Last known gateway:

- ???
  gateway aliases
  - none

Broadcast/location port: 9494

Additional gateway location servers:

- 9.3.1.133+9494
- 9.3.1.210+9494

Additional configuration options:

```
-g grizzly+9494
```

Figure 53. Location Configuration Panel

In the above example, we are attempting to modify the login_interfaces information of the Endpoint because both Endpoint gateways that are displayed in the Additional gateway location servers field are unreachable. To add the Endpoint gateway, `grizzly` (9.3.1.134), into the login_interfaces via the Endpoint Web interface, you will need a user ID and password. By default, the user ID is `tivoli` and the initial password until completion of the initial login is `boss`. After the completion of the initial login, the password can be obtained by the `wep` command.

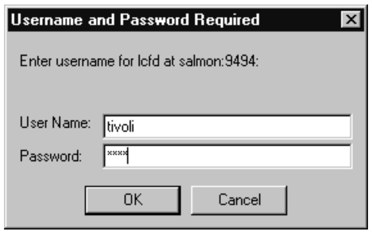
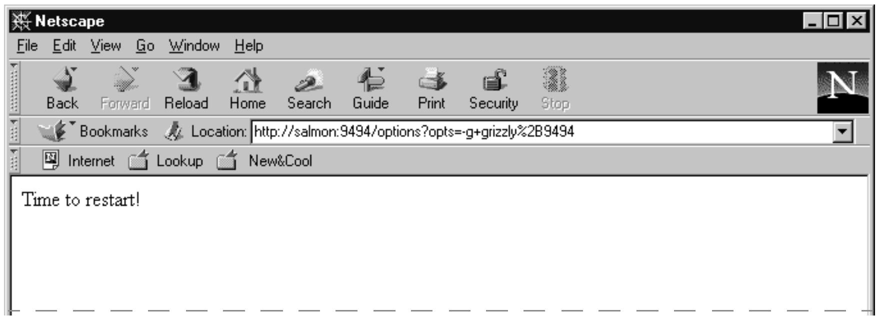

Username and Password Required
Enter username for lcfd at salmon:9494:
User Name: tivoli
Password: ****
OK
Cancel

Figure 54. Password Dialog

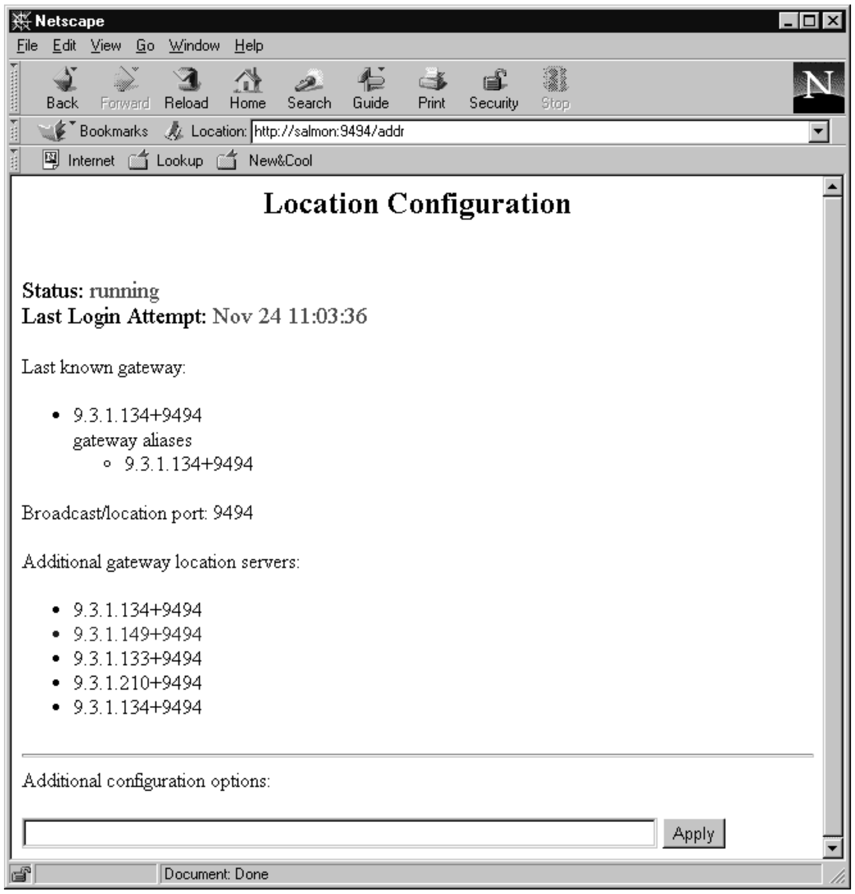
Enter the appropriate user ID and password in the dialog box and push **OK** The Endpoint will be restarted with the modified configurations. The restart message will be shown as follows.


Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://salmon:9494/options?opts=-g+grizzly%2B9494
Internet Lookup New&Cool

Time to restart!

Figure 55. Restart Message

After this operation, go back to the Local Configuration page and check the status field. It should be shown as running. This means the Endpoint has logged into the Endpoint gateway which is shown in the Last known gateway field successfully. In this case, we defined the select_gateway_policy and it returnd some candidates, so that the Additional gateway location server shows some alternate gateways (refer to the Figure 56 on page 149).


Netscape
File Edit View Go Window Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://salmon:9494/addr
Internet Lookup New&Cool
Apply
Document: Done

Location Configuration

Status: running
Last Login Attempt: Nov 24 11:03:36

Last known gateway:

- 9.3.1.134+9494
  gateway aliases
  - 9.3.1.134+9494

Broadcast/location port: 9494

Additional gateway location servers:

- 9.3.1.134+9494
- 9.3.1.149+9494
- 9.3.1.133+9494
- 9.3.1.210+9494
- 9.3.1.134+9494

Additional configuration options:

Figure 56. Local Configuration Panel after Rescue

The summary of the test follows.

Table 20. Case 8: The Endpoint Web Interface Test

Case 8	
Scenario	Modify the login interfaces list of the Endpoint using the Endpoint Web interface. In this case, the Endpoint has not completed the initial login yet.
Option	`lcfd -g grizzly+9494`
EP Policy	`select_gateway_policy (grizzly, yi2250d)`
Result	The login interfaces list was modified by the Web interface and the Endpoint can log into the Endpoint gateway.

5.3.7 TMR Redirection

TMR redirection is one of the most effective solutions for managing very large environments containing more than one TMR. In the previous chapter (Chapter 4, “Configuring the TMA Environment” on page 105), we introduced TMR redirection, and we will describe TMA behavior when we configured TMR redirection configuration in our environment.

5.3.7.1 Interconnected TMRs

First of all, we need to configure interconnected TMRs in our test environment. It is quite easy and can be configured from the Desktop interface. After configuring the TMR connections, we can see all managed resources belonging to both TMRs in the Top Level Policy Regions panel, as shown in Figure 57.

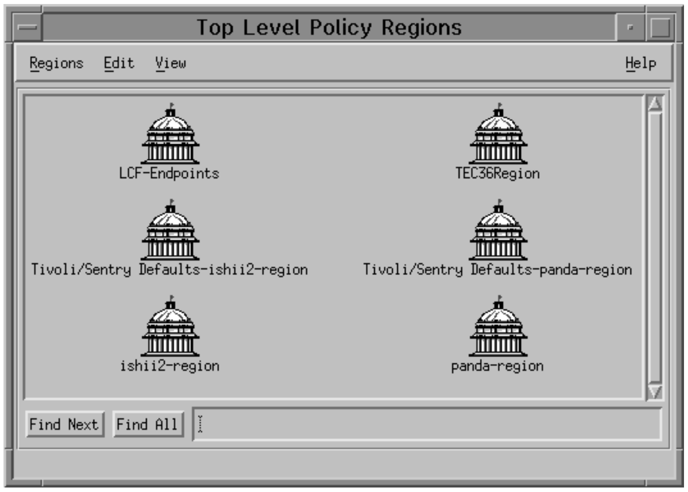


Figure 57. The Top Level Policy Regions

Note

TMR redirection can be configured with both one-way and two-way connection. You can specify the TMR connection type, one-way or two-way, on the panel configuring the TMR connection.

5.3.7.2 Understanding TMR Redirection

Now we have configured the TMR connections environment. We will now introduce the result of the our TMR redirection test.

Case 9:

In this case, we introduce the TMR redirection example. Before testing, we connected one TMR to another TMR with one-way connection. We also defined the select_gateway_policy in TMR-A for redirecting the TMR when the Endpoint would attempt to perform the login. Then we started to install the `lcfd` daemon for performing the initial login. We will show more detail in Figure 58 on page 152.

Table 21. Case 9: TMR Redirection Test

Case 9	
Scenario	Before testing, configure the multiple TMRs and connect TMRs.The select_gateway_policy for redirecting TMR should also be defined before this test. Then install `lcfd` with the following options for performing the initial login in the multiple TMR environment.

Option	`-g kodiak+9494`
	`-D bcast_disable=1`

EP Policy	`select_gateway_policy (trout)`
Result	The Endpoint attempted to perform the initial login to the EP gateway specified by the -g option . The EP Manager referred to the select_gateway_policy defined in the TMR-A and found the EP gateway in the TMR-B. The EP Manager returned the login information for redirecting to the Endpoint. As a result, the Endpoint logged in to the EP gateway in the TMR-B. The TMR redirection completed successfully.

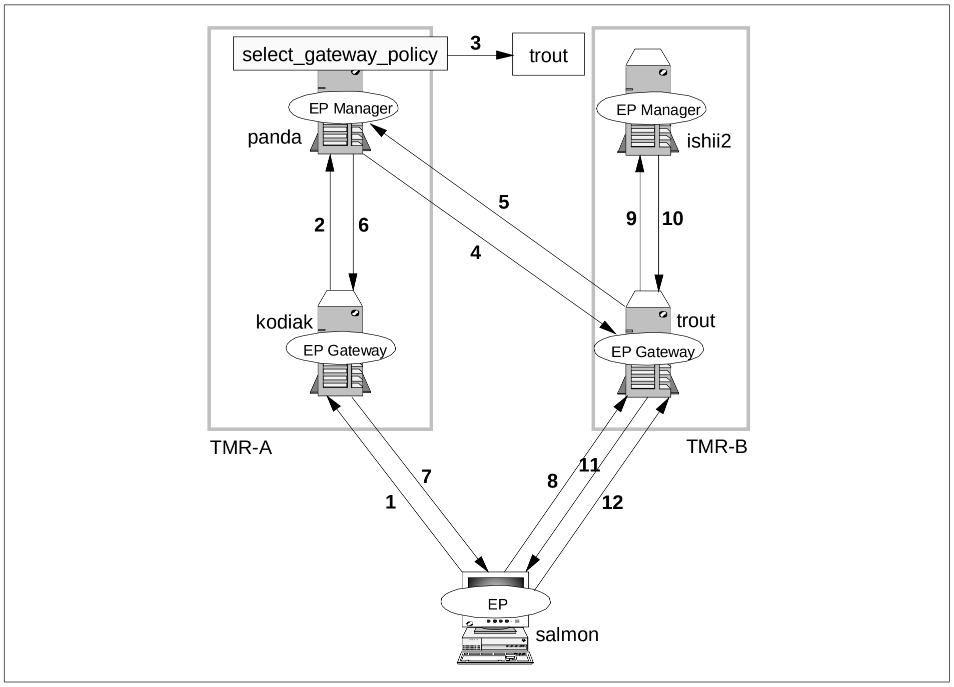


Figure 58. The TMR Redirection

1. The Endpoint (salmon) attempted to perform the initial login to the Endpoint gateway (kodiak) specified first by the -g option.
2. The Endpoint gateway (kodiak) forwarded the login request to the Endpoint Manager (panda).
3. The Endpoint Manager (panda) received the login request from the Endpoint (salmon), then referred to the select_gateway_policy defined in TMR-A and got the candidate for the assigned gateway. The Endpoint gateway (trout) that the Endpoint Manager (panda) found belonged to TMR-B, so that the Endpoint Manager (panda) then recognized this login process as TMR redirection.
4. The Endpoint Manager (panda) sent the request for retrieving the interface information of the Endpoint gateway (trout).
5. The Endpoint Manager (panda) retrieved the Endpoint gateway's (trout) interface information.
6. The Endpoint Manager (panda) sent the network interface information of the Endpoint gateway (trout) with the directions to use it as the intercepting gateway (trout) to the original intercepting gateway (kodiak).

7. The Endpoint gateway (kodiak) relayed the information to the Endpoint (salmon).
8. The Endpoint (salmon) also recognized the login request was redirected and began the initial login process again with the new information. The login request was intercepted by the Endpoint gateway (trout), as designated in the select_gateway_policy script in the TMR-A.
9. The login request was forwarded to the Endpoint Manager (panda) in TMR-B.
10. Having no defined Endpoint policy in TMR-B, the Endpoint Manager (panda) assigned the Endpoint (salmon) to the intercepting gateway (trout) and sent the new login information to the intercepting gateway (trout).
11. The intercepting gateway (trout) relayed the login information to the Endpoint (salmon).
12. As a result, the Endpoint logged in to its assigned gateway (trout).

After you configure TMR redirection, the Endpoint performs the login to the Endpoint gateway and then the Endpoint is redirected by the Endpoint policy. The following messages appear in the lcfd.log file. Look at line 9 of the file. The Endpoint recognizes TMR redirection at that time.

```
Dec 09 11:26:50 Q lcfd send_login_dgram: interval=30 attempts=2
Dec 09 11:26:50 Q lcfd net_usend of 346 bytes to 9.3.1.133+9494.  Bcast=0
Dec 09 11:26:50 Q lcfd send_login_dgram: waiting for reply. attempt 1 of 2
Dec 09 11:26:50 Q lcfd net_accept, handle=0x305f60
Dec 09 11:26:57 Q lcfd New connection from 9.3.1.133+2153
Dec 09 11:26:57 Q lcfd Entering net_recv, receive a message
Dec 09 11:26:57 Q lcfd Leaving net_recv: bytes=346, (type=14 session=0)
Dec 09 11:26:57 Q lcfd recv: len='346' (code='14', session='0')
Dec 09 11:26:57 1 lcfd tmr redirect
Dec 09 11:26:57 2 lcfd Trying other login listeners...
Dec 09 11:26:57 Q lcfd send_login_dgram: interval=30 attempts=2
Dec 09 11:26:57 Q lcfd net_usend of 346 bytes to 9.3.1.210+9494.  Bcast=0
Dec 09 11:26:57 Q lcfd send_login_dgram: waiting for reply. attempt 1 of 2
Dec 09 11:26:57 Q lcfd net_accept, handle=0x305f60
Dec 09 11:26:59 Q lcfd New connection from 9.3.1.210+33582
Dec 09 11:26:59 Q lcfd Entering net_recv, receive a message
Dec 09 11:26:59 Q lcfd Leaving net_recv: bytes=498, (type=14 session=0)
Dec 09 11:26:59 Q lcfd recv: len='498' (code='14', session='0')
Dec 09 11:26:59 2 lcfd Writing GCS file: C:\Tivoli\lcf\dat\1\last.cfg
Dec 09 11:26:59 1 lcfd salmon.117 is dispatcher 117 in region 1588251808
Dec 09 11:26:59 1 lcfd write login file 'lcf.dat' complete
Dec 09 11:26:59 1 lcfd Logging into new gateway...
Dec 09 11:26:59 Q lcfd login_to_gw
Dec 09 11:26:59 Q lcfd login_gw -> 9.3.1.210+9494
Dec 09 11:26:59 2 lcfd Connecting to '9.3.1.210+9494'
Dec 09 11:26:59 Q lcfd net_send of 506 bytes, session 117
Dec 09 11:26:59 Q lcfd net_accept, handle=0x305f60
Dec 09 11:26:59 Q lcfd New connection from 9.3.1.210+33583
Dec 09 11:26:59 Q lcfd Entering net_recv, receive a message
Dec 09 11:26:59 Q lcfd Leaving net_recv: bytes=514, (type=14 session=117)
Dec 09 11:26:59 Q lcfd recv: len='514' (code='14', session='117')
Dec 09 11:26:59 2 lcfd Writing GCS file: C:\Tivoli\lcf\dat\1\last.cfg
Dec 09 11:26:59 1 lcfd salmon.117 is dispatcher 117 in region 1588251808
Dec 09 11:26:59 1 lcfd write login file 'lcf.dat' complete
Dec 09 11:26:59 1 lcfd final pid: 120
Dec 09 11:26:59 1 lcfd Login to gateway 9.3.1.210+9494 complete.
```

5.4 Understanding TMA Behavior in Unexpected Situations

In real environments, the network may sometimes go down or the TMR server may be unavailable. Then what happens? In this section, we assume these situations and perform tests regarding them. We analyze the results and explain the behavior of the TMA.

5.4.1 Understanding Initial Login

In this section, we introduce the Endpoint behavior in an unexpected situation during the Endpoint initial login.

5.4.1.1 Endpoint Manager is Unavailable during Initial Login

The LCF architecture reduces the TMR Server’s workload, but the TMR Server is still very important in the Tivoli Management environment. In this test, we assume the TMR Server becomes unavailable.

Case 10:

In this test, we performed a shutdown of the Endpoint Manager (panda) before testing. Then we started to install the `lcfd` with the following options.

The `lcfd` attempted to perform the initial login to the Endpoint gateways specified by the -g option. However, the Endpoint gateways that received login requests from the Endpoint couldn't receive any response from the Endpoint Manager even if the gateway forwarded the request to the Endpoint Manager, so the initial login failed. As a result, the Endpoint couldn't log in to any Endpoint gateways.

Table 22. Case 10: The Initial Login when EP Manager is Unavailable

Case 10	
Scenario	Shutdown the EP Manager, then install `lcfd` for performing the initial login.

Option	`lcfd -g trout+9494:kodiak+9494`
	`-D bcast_disable=1`

EP Policy	None
Result	The Endpoint couldn't log in to any EP gateways.

The following figure (Figure 59 on page 155) shows this situation.

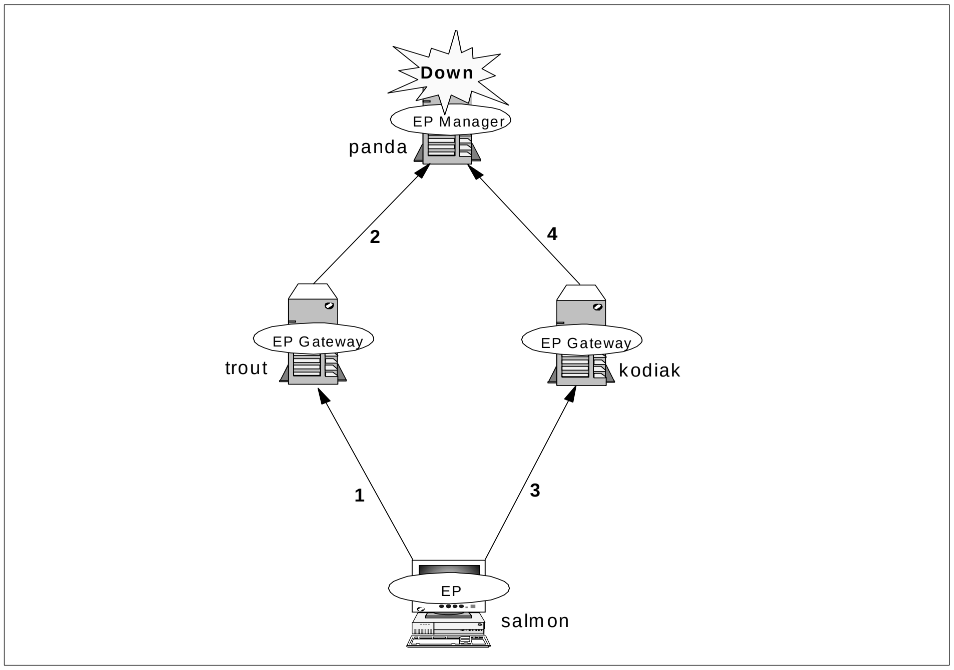


Figure 59. The Initial Login when the EP Manager is Unavailable

1. The Endpoint (salmon) attempted to perform the initial login to the Endpoint gateway (trout) specified by the -g option as the first Endpoint gateway in the login interfaces list. Trout received the login request from salmon. The Endpoint (salmon) sent the login request to the Endpoint gateway (trout) six times every five minutes by default. This depends on the udp_attempts and udp_interval parameters. The login request is a UDP packet, so the Endpoint just sends the login request using UDP and waits for a response from the Endpoint gateway. The Endpoint doesn’t establish a TCP session between the Endpoint and Endpoint gateway in this case.
2. The Endpoint gateway (trout) forwarded the login request from salmon to the Endpoint Manager (panda). This is also a UDP packet. However, the Endpoint (salmon) couldn’t get any response from the Endpoint Manager (panda) because the Endpoint Manager (panda) was not available. Actually, the login request from the Endpoint (salmon) would be forwarded to the Endpoint Manager (panda) everytime the Endpoint gateway received the login request from the Endpoint.
3. After sending the login requests (six times every five minutes), salmon gave up logging in to trout. Then salmon attempted to perform the initial login to the second Endpoint gateway (kodiak) specified by the -g option. This process is the same as the process shown in Step 1.
4. The Endpoint gateway (kodiak) forwarded the login request from salmon to the Endpoint Manager (panda) every time kodiak received the login request. Since the Endpoint Manager was down, nothing happened. After sending the login requests (six times every five minutes), the Endpoint attempts to perform a broadcast to look for an Endpoint gateway if we don’t define `bcast_disable=1` (broadcast is disabled). In this case, we disabled the broadcast (`bcast_disable=1`), so that the Endpoint will wait for a while (20 minutes by default, but it depends on the login_interval parameter) and retry the initial login to the first Endpoint gateway (trout). This means these processes (process 1 - 4) will be continued indefinitely as long as the Endpoint Manager is unavailable.

5.4.1.2 Endpoint Gateway is Unavailable at Initial Login

The Endpoint gateway plays a vital role in the TMA environment. Basically, all requests issued from the Endpoint would be handled by the Endpoint gateway, so that at least one Endpoint gateway must be available in the TMR. But, the relationship between the Endpoint gateway and Endpoint is really flexible, as you know, and it is completely different from the relationship between the TMR server and the Managed Node. In this, we assume the Endpoint gateway where the Endpoint attempts to log in becomes unavailable. Since this sort of situation is fairly common, you should understand the behavior of TMA in this case.

Case 11:
In this test, we specified the alternate Endpoint gateway using the -g option. Before testing, we made the Endpoint gateway (grizzly) where the Endpoint would attempt to perform the initial unavailable login. As a result, the Endpoint gave up attempting to perform the initial login to the first Endpoint gateway (grizzly), then attempted to perform the initial login to the alternate Endpoint gateway (kodiak), where no Endpoint policy has been set.

Table 23. Case 11: The Initial Login when EP Gateway is Unavailable

Case 11	
Scenario	Specify the unavailable EP gateway (grizzly) and available EP gateway (kodiak) as follows.
Option	`lcfd -g grizzly+9494:kodiak+9494`
EP Policy	None
Result	First of all, the EP attempted to perform initial login to the first EP gateway (grizzly) specified by the -g option, but it was not available. It then attempted to perform initial login to the second EP gateway (kodiak) and logged in successfully.

The following figure, Figure 60 on page 158, displays the Endpoint login process in this case.

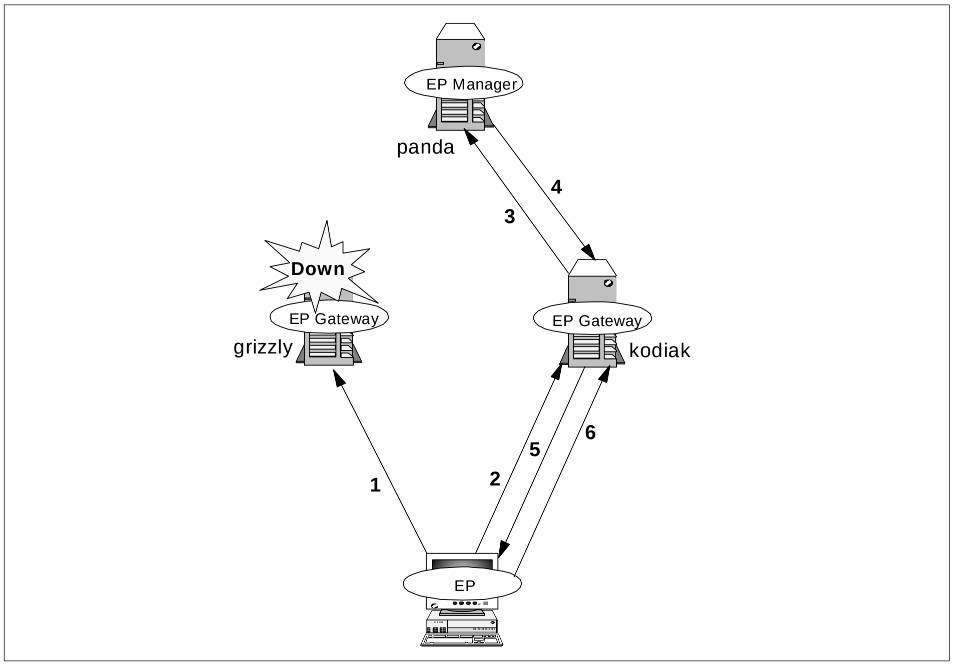


Figure 60. Endpoint Initial Login to the Alternate Endpoint Gateway

1. The Endpoint attempts to perform the initial login to the Endpoint gateway (grizzly) at first. The Endpoint tried again to send the initial login request to grizzly until the timeout. By default, timeout is 30 minutes (six times every five minutes) but you can change the timeout using the udp_attempts and udp_interval parameters. Then the Endpoint gives up sending the initial login request to grizzly.
2. After the timeout, the Endpoint attempts to perform the initial login to the alternate Endpoint gateway (kodiak).
3. The Endpoint gateway (kodiak) forwards the login request to the Endpoint Manager (panda).
4. Having no defined policy, panda assigns the Endpoint to kodiak and sends the new login information to kodiak.
5. Kodiak relays the login information to the Endpoint.
6. The Endpoint logs in to its assigned gateway, kodiak.

Case 12:
This case is similar to Case 11. The only difference is we didn't specify the alternate Endpoint gateway. As a result, the Endpoint attempted to broadcast

after the Endpoint initial login failure to the specified Endpoint gateway (grizzly) using the -g option.

Table 24. Case 12: The Initial Login when EP Gateway is Unavailable

Case 12	
Scenario	Specify only the unavailable EP gateway (grizzly) with the -g option.
Option	`lcfd -g grizzly+9494`
EP Policy	None
Result	First of all, the EP attempted to perform initial login to the EP gateway (grizzly) specified by the -g option but it was not available. It then attempted to perform initial login using broadcast.

5.4.1.3 Unexpected Situation during Gateway Selection

This case is both interesting and complicated, however, it is a good example for understanding the Endpoint initial login. In this case, we defined the select_gateway_policy for assigning the Endpoint gateway to the Endpoint.

Case 13:

This case helps us to understand how to select the assigned gateway. We defined the select_gateway_policy that returns two Endpoint gateways as the candidates for the assigned gateway. In this policy, the first candidate is grizzly and the second candidate is kodiak. We performed a shutdown of grizzly, which is the first candidate for the assigned gateway. Then we started to install the `lcfd` with the options shown in Table 25 on page 159 for performing the initial login. We will provide more detail in Figure 61.

Table 25. Case 13: An Unexpected Situation during EP Gateway Selection

Case 13	
Scenario	Define the select_gateway_policy that returns grizzly as the first assigned EP gateway and kodiak as the second assigned EP gateway before testing. We also perform shutdown grizzly before testing. Then we install the `lcfd` for performing the initial login.

Option	`lcfd -g trout+9494`
	`-D bcast_disable=1`

EP Policy	`select_gateway_policy (grizzly, kodiak)`
Result	The available EP gateway was assigned to the Endpoint by the select_gateway_policy and the Endpoint logged in to the available EP gateway.

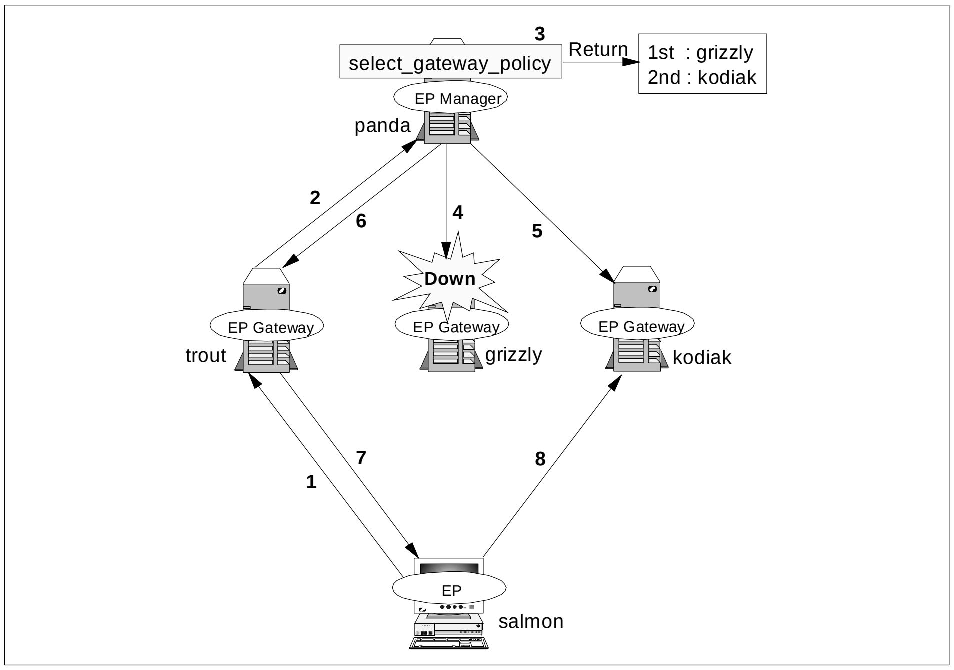


Figure 61. One of the Selected Gateways in the Policy is Not Available

1. The Endpoint (salmon) sent the initial login request to the Endpoint gateway (trout) specified by the -g option. In this case, trout was the intercepting gateway. The intercepting gateway (trout) received the login request from salmon.
2. Trout forwarded the login request to the Endpoint Manager (panda).
3. The Endpoint Manager (panda) referred to the select_gateway_policy and received two candidates for the assigned gateway.
4. Panda attempted to connect to grizzly, which was the first candidate for the assigned gateway using the `new_endpoint` method. But the connection with grizzly failed.
5. Then panda attempted to connect to kodiak, which was the second candidate for the assigned gateway using the `new_endpoint` method and the Endpoint gateway (kodiak) was connected successfully.
6. As a result, panda assigned the Endpoint to kodiak as the assigned gateway and returned the login assignment information to the intercepting gateway (trout).

7. The intercepting gateway (trout) then relayed the login information to the Endpoint (salmon).
8. The Endpoint (salmon) logged in to the assigned gateway (kodiak). This process is the same as a normal login to kodiak.

Case 14:
This case is similar to Case 13, but we shutdown both Endpoint gateways that were configured as the first candidate and second candidate for the assigned gateway in the select_gateway_policy. Therefore, there is no available Endpoint gateway configured as the candidate for the assigned gateway in the select_gateway_policy. We will explain how the Endpoint Manager handled this situation in detail in the next figure, Figure 62 on page 162.

Table 26. Case 14: An Unexpected Situation during EP Gateway Selection

Case 14	
Scenario	Define the select_gateway_policy that returns grizzly as the first assigned EP gateway and kodiak as the second assigned EP gateway before testing. We also perform shutdown grizzly and kodiak before testing. Then we install the `lcfd` for performing the initial login.

Option	`lcfd -g trout+9494`
	`-D bcast_disable=1`

EP Policy	`select_gateway_policy (grizzly, kodiak)`
Result	Since there was no available EP gateway that configured in the select_gateway_policy, the Endpoint Manager assigned the intercepting gateway as the assigned gateway, so that the Endpoint logged in to the intercepting gateway.

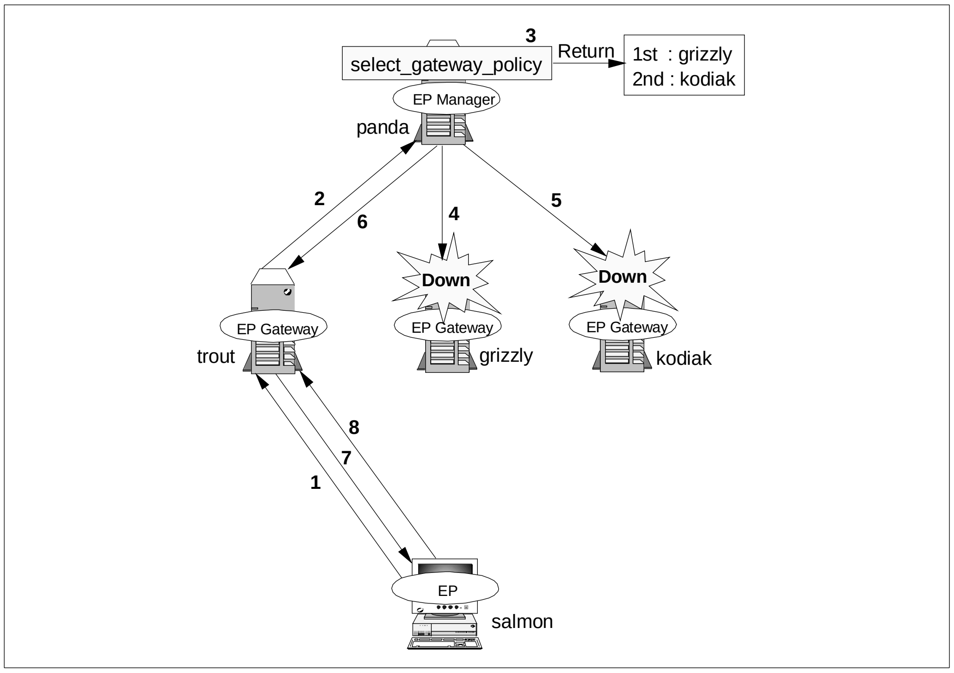


Figure 62. There is No Available Gateway in the select_gateway_policy

1. The Endpoint (salmon) sent the initial login request to the Endpoint gateway (trout) specified by the -g option. In this case, trout was the intercepting gateway. Then trout received the login request from salmon.
2. Trout forwarded the login request to the Endpoint Manager (panda).
3. The Endpoint Manager (panda) referred to the select_gateway_policy and got the two candidates for the assigned gateway.
4. Panda attempted to connect to grizzly which was the first candidate for the assigned gateway, by first using the `new_endpoint` method. But, the connection with grizzly failed.
5. Then panda attempted to connect to kodiak, which was the second candidate for the assigned gateway using the `new_endpoint` method but it also failed.
6. As a result, panda assigned the Endpoint (salmon) to trout as the assigned gateway and returned the login assignment information to the intercepting gateway (trout).
7. Then the intercepting gateway (trout) relayed the login information to the Endpoint (salmon).

8. Finally, the Endpoint (salmon) logged in to the intercepting gateway (trout). This process is the same as a normal login to trout.

Note

The Endpoint Manager actually attempts to contact all Endpoint gateways that are defined in the select_gateway_policy, even if the Endpoint Manager detects the first candidate is available. As a result, the unreachable Endpoint gateways at the gateway selection process does not appear in the login interfaces list.

The Endpoint Manager uses the new_endpoint method to contact the Endpoint gateway. This means that the Endpoint Manager recognizes what is unavailable when the Endpoint gateway process (gateway) is stopped, even if the Endpoint gateway machine is running fine.

5.4.2 Understanding Normal Login with Unexpected Situations

In this section, we introduce the Endpoint behavior in an unexpected situation during the Endpoint normal login.

5.4.2.1 Endpoint Manager is Unavailable in Normal Login

We assume the Endpoint Manager is not available when the Endpoint attempts to perform the normal login. Then, we introduce an example in order to understand the Endpoint behavior in these situations.

Case 15: In this case, we performed a shutdown of the Endpoint Manager before testing. Then we re-started the `lcfd` with the following option for performing the normal login.

Table 27. Case 15: The Normal Login when the EP Manager is Unavailable

Case 15	
Scenario	Shutdown the EP Manager, then perform the normal login.
Option	`lcfd -g trout+9494:kodiak+9494` `-D dcast_disable=1`
EP Policy	None
Result	The Endpoint logged in to the assigned gateway (trout).

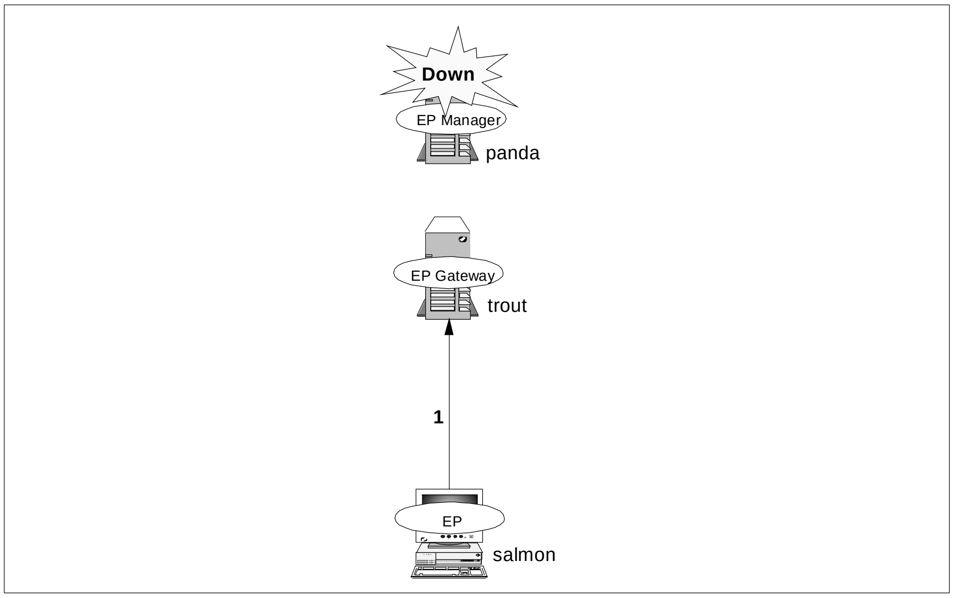


Figure 63. The Normal Login when EP Manager is Unavailable

1. The Endpoint (salmon) logged in to the assigned gateway (trout).

Note

In this case, the Endpoint could log in to the assigned gateway without the Endpoint Manager. However the `boot_method` had not been executed yet. To execute the `boot_method`, the Endpoint Manager should be available. As a result, the method should be booted from the `boot_method` because the Sentry engine will not be started. You can confirm it using the gatelog file in the Endpoint gateway. When the Endpoint Manager is available, the following messages will be witten to the gatelog file normally:

```
1998/11/24 17:24:33 +06: login succeeded for 9.3.1.193+1399
(salmon,w32-ix86,RF+7TTDM068DSHCR6Q4700000537,reg=1189622596,od=163)

1998/11/24 17:24:33 +06: run_login_policy: Running login policy on endpoint
salmon.

1998/11/24 17:24:36 +06: run_ep_boot_methods: nothing to do.
```

However, when the Endpoint Manager is not available, as in this case, the `run_ep_boot_methods` method would not be invoked, so that the third line of the above list would not be written to the gatelog file.

5.4.2.2 Understanding Endpoint Isolation

When the Endpoint attempts to perform the normal login to its assigned gateway, the gateway may be unreachable. In this case, the Endpoint is said to be isolated. We assume this situation and monitor what happens.

Case 16: To isolate the Endpoint, we stopped the Endpoint gateway after the initial login had been completed. Then we re-started the `lcfd` for performing the normal login to the Endpoint gateway we stopped. We will explain how the Endpoint recovered from the isolated situation in Figure 64 on page 166.

Table 28. Case 16: The Endpoint Isolation

Case 16	
Scenario	Perform the shutdown the EP gateway after the initial login is completed for making the Endpoint isolated. Then we restart the Endpoint and the Endpoint attemps to perform the normal login.

Option	`lcfd -g kodiak+9494`
	`-D bcast_disable=1`

EP Policy	`select_gateway_policy (kodiak, trout)`
Result	The Endpoint attempted to perform the normal login to the assigned gateway, however, kodiak was not available. Therefore, the Endpoint has an isolated status. Then the Endpoint attempted to perform the isolated login to the EP gateway specified in the login interfaces list in the lcf.dat file and sent the login request to the EP gateway. The select_gateway_policy was invoked during the login process and returned the assigned gateway. As a result, the Endpoint logged in to the EP gateway assigned by the select_gateway_policy.

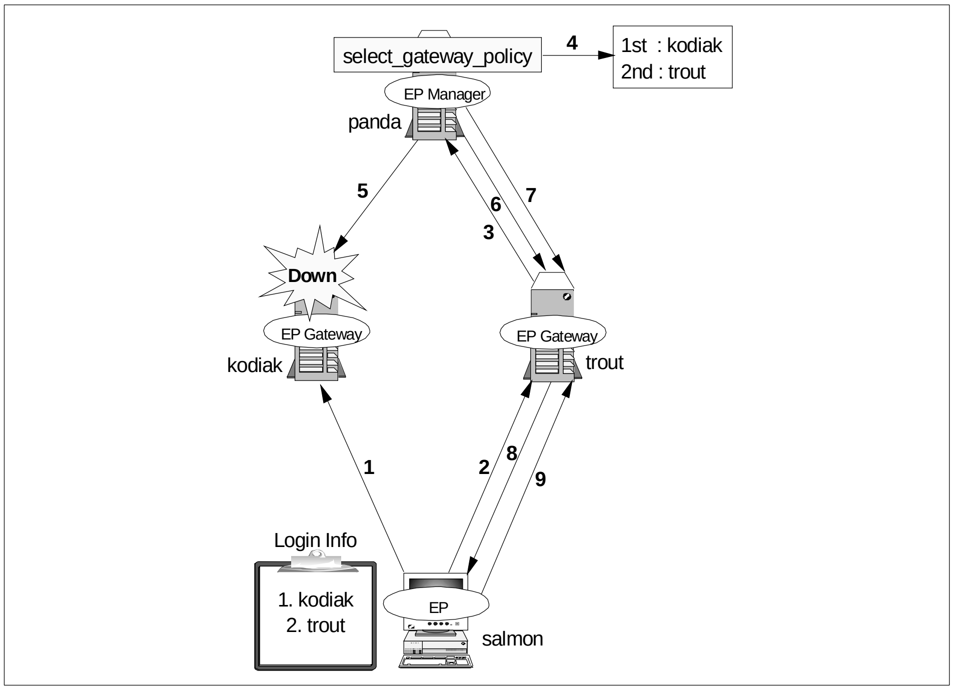


Figure 64. The Recovery from Isolated Situation

1. The Endpoint (salmon) attempted to perform the normal login to the assigned gateway (kodiak). However, the assigned gateway (kodiak) was not available. The Endpoint (salmon) attempted to perform the normal login again until the timeout occurred.
2. After the timeout, the Endpoint (salmon) was considered isolated. Then the Endpoint (salmon) attempted to send the login request to the Endpoint gateway (trout), which is contained in the login interfaces list of the lcf.dat file.
3. The Endpoint gateway (trout) forwarded the login request to the Endpoint Manager (panda).
4. The Endpoint Manager (panda) received the login request from the Endpoint (salmon) and referred to the select_gateway_policy. It got two candidates for the assigned gateway.
5. The Endpoint Manager (panda) attempted to contact the Endpoint gateway (kodiak), which was the first candidate for the assigned gateway by first using the `new_endpoint` method. However, the connection with the gateway (kodiak) failed.

6. Then the Endpoint Manager attempted to connect to the Endpoint gateway (trout), which was the second candidate for the assigned gateway using the `new_endpoint` method and the Endpoint gateway (trout) was connected successfully.
7. As a result, the Endpoint Manager (panda) assigned the Endpoint (salmon) to the Endpoint gateway (trout) as the assigned gateway and returned the login assignment information to the intercepting gateway (trout).
8. The intercepting gateway (trout) then relayed the login information to the Endpoint (salmon).
9. The Endpoint (salmon) logged in to the assigned gateway (trout).

As you can see from the above flow, the isolated Endpoint is normally migrated automatically to another Endpoint gateway. The following message appears in the gatelog file during the migration process.

```
1998/11/24 17:24:29 +06: eplogin (0): forwarding isolation login(2)to epmgr
```

This message will be logged only when the isolated Endpoint attempts to send the login request.

Case 17:
This case was similar to Case 16, but we defined the broadcast as enabled (`bcast_disable=0`) and performed a shutdown to the alternate Gateway (trout), where no Endpoint policy has been set. Detailed information about this is shown in the next figure, Figure 65.

Table 29. Case 17: The Endpoint Isolation

Case 17	
Scenario	Shutdown the assigned gateway and alternate gateway, then perform the normal login.

Option	`lcfd -g kodiak+9494`
	`-D bcast_disable=0`

EP Policy	None
Result	The Endpoint was isolated. Then the Endpoint attempted to send the login request to the alternate gateway gateway, but the alternate gateway was not available as well. Then the Endpoint attempted to perform the broadcast for the initial login and logged in to the intercepting gateway.

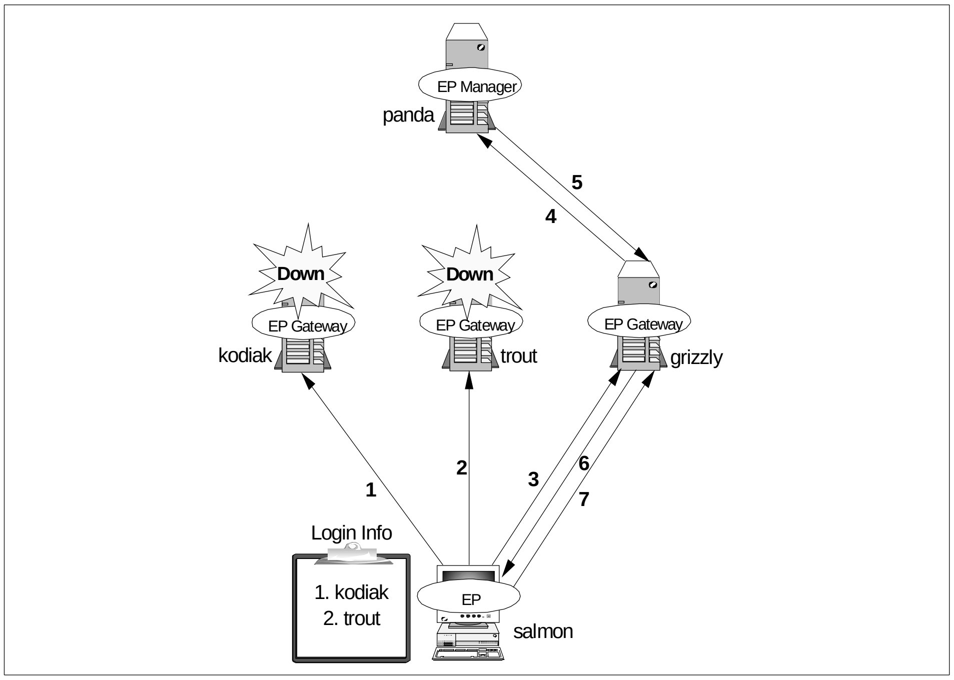


Figure 65. The Recovery from the Isolated Situation Using Broadcast

1. The Endpoint (salmon) attempted to perform the normal login to the assigned gateway (kodiak). However, the assigned gateway (kodiak) was not available. The Endpoint (salmon) tried again to perform the normal login until the timeout occurred.
2. After the timeout, the Endpoint (salmon) was considered isolated. Then the Endpoint (salmon) attempted to send the login request to the alternate gateway (trout) contained in the login interfaces list of the lcf.dat file. However, the Endpoint couldn't receive any response from the Endpoint gateway (trout). Sending the login request continues until the timeout occurs.
3. After the timeout, the Endpoint (salmon) attempted to perform the broadcast for the initial login. Then the login request was intercepted by the Endpoint gateway (grizzly).
4. The Endpoint gateway (grizzly) forwarded the login request to the Endpoint Manager (panda).
5. Having no defined Endpoint policy, the Endpoint Manager (panda) assigned the Endpoint (salmon) to the intercepting gateway (grizzly) and sent the new login information to the Endpoint gateway (grizzly).

6. The Endpoint gateway (grizzly) relayed the login information to the Endpoint (salmon).
7. The isolated Endpoint (salmon) logged in to its assigned gateway (grizzly).

Case 18:
This case was also similar to Case 16, but we shutdown all alternate gateways and no Endpoint policy has been set. Detailed information about this is shown in the next figure.

Table 30. Case 18: The Endpoint Isolation

Case 18	
Scenario	Shutdown the assigned gateway and alternate gateway, then perform the normal login.
Option	lcfd -g kodiak+9494 -D bcast_disable=1
EP Policy	None
Result	The Endpoint was isolated. Then the Endpoint attempted to send the login request to the EP gateways that were contained in the login interfaces list but no gateway was available. As a result, the isolated Endpoint couldn’t log in to any EP gateway.

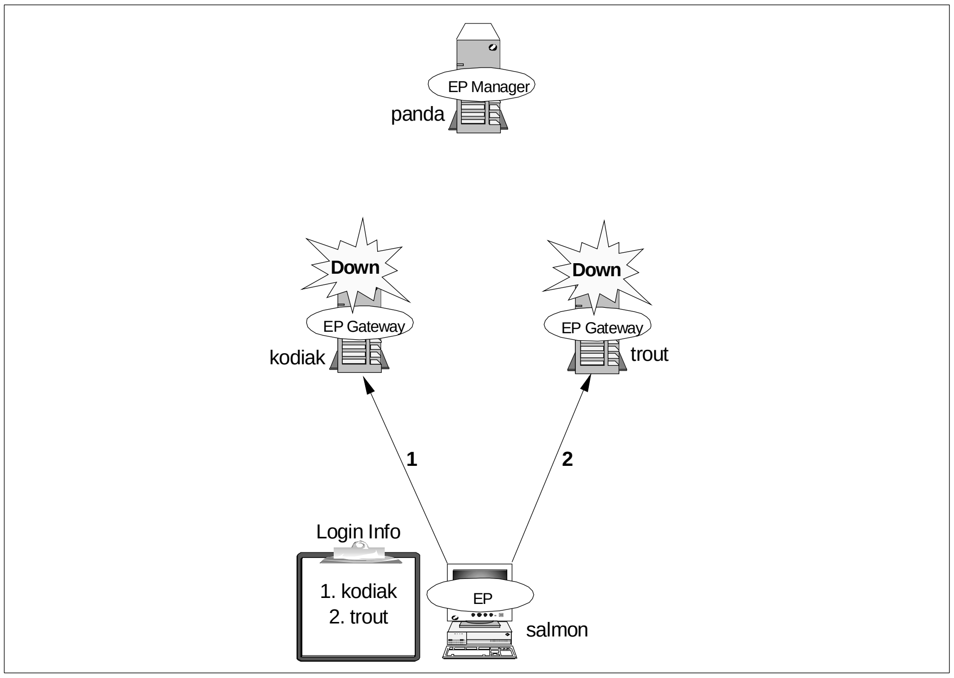


Figure 66. The Isolated Endpoint

1. The Endpoint (salmon) attempted to perform the normal login to the assigned gateway (kodiak). However, the assigned gateway (kodiak) was not available. The Endpoint (salmon) tried again to perform the normal login until the timeout occurred.
2. After the timeout, the Endpoint (salmon) was considered isolated. Then the Endpoint (salmon) attempted to send the login request to the alternate gateway (trout) that was contained in the login interfaces list in the lcf.dat file. But the Endpoint gateway (trout) was also unavailable. The Endpoint (salmon) tried again to send the login request until the timeout occurred. After the timeout, the Endpoint would try again to perform processes 1 and 2 every 20 minutes (login_interval).

Note

Once the initial login is complete, no more initial logins occur. There are differences between the initial and isolated logins:

- The allow_install_policy and after_install_policy only run on the initial login.
- The communications are encrypted with the private key on non-initial logins.

5.4.2.3 Unexpected Situation after Endpoint Login Completion

Sometimes the Endpoint Manager or Endpoint gateway may go down, or the network between the Endpoint Manager and Endpoint gateway or the Endpoint gateway and Endpoint may go down. In this section, we assume these situations occurred and verify the behavior of the TMA in these situations. This result is very useful for designing the TMR and planning the allocation of resources in your environment.

Case 19 and 20:

These cases assumes the Endpoint gateway becomes unreachable from the Endpoint after the Endpoint login succeeded. First of all, we configured the TMA environment in this case. This means the Endpoint logged in to the Endpoint gateway successfully. Then we disconnected the network between the Endpoint gateway and the Endpoint. In this situation, we issued an upcall/downcall. We used our sample programs (refer to Appendix 8, “Overview of TMA Internals and Application Development” on page 281 for details) to issue the upcall/downcall from the CLI interface in this test. We show the behavior of the TMA when we issued the upcall in the next figure.

Table 31. Case 19 and 20: Network Becomes Unavailable after Login Completion

Case 19 and 20	
Scenario	After the Endpoint login succeeded, we disconnect the network between the EP gateway and Endpoint. Then we issue the upcall/downcall.

Option	`lcfd -g grizzly+9494`
	`-D bcast_disable=1`

EP Policy	None

Case 19 and 20	
Result	***Issuing the upcall:*** The upcall was issued from the Endpoint, but the Endpoint couldn't receive any response from the EP gateway. Then the Endpoint understood the assigned gateway was unavailable. The Endpoint attempted to perform the isolated login to the alternate gateway. As a result, the Endpoint recovered from the situation automatically.
	Issuing the downcall: When any downcall is invoked, the EP gateway must be available, so the downcall can't be invoked in this case. The downcall failed.

The following figure shows the TMA behavior when we invoked the upcall program in this case.

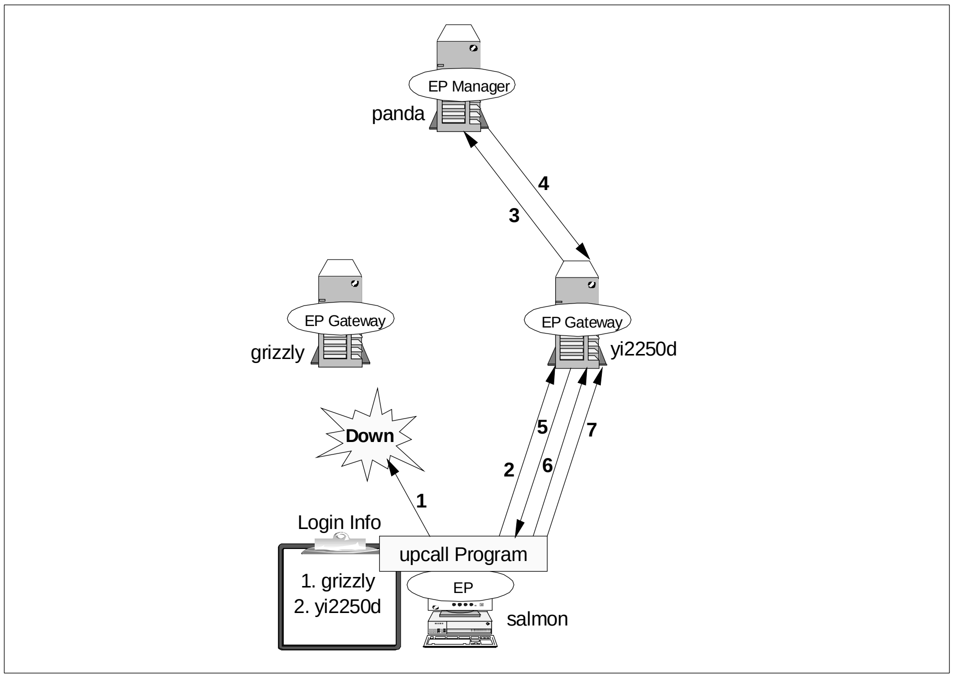


Figure 67. Issuing Upcall when the EP Gateway is Unreachable

1. Before the testing, the Endpoint (salmon) had logged in to the assigned Gateway (grizzly) successfully. Then we disconnected the network between the assigned gateway (grizzly) and the Endpoint (salmon). In this situation, we invoked the upcall-generating program to issue the upcall to the assigned gateway (grizzly). However, the assigned gateway (grizzly) was not available at that time. The Endpoint (salmon) couldn't get

any response from the assigned gateway (grizzly), so the Endpoint (salmon) understood the assigned gateway (grizzly) went down. The Endpoint (salmon) attempted to perform the login to the assigned gateway (grizzly) to make sure of the assigned gateway's status, but it failed.

2. Then the Endpoint (salmon) attempted to perform the initial login to the alternate gateway (yi2250d) that was contained in the login information of the lcf.dat file. The Endpoint (salmon) sent the initial login request to the alternate gateway (yi2250d).
3. The alternate gateway (yi2250d) forwarded the login request to the Endpoint Manager (panda).
4. Having no defined policy, the Endpoint Manager (panda) assigned the Endpoint (salmon) to the alternate gateway (yi2250d) and sent the new login information to the Endpoint gateway (yi2250d).
5. The Endpoint gateway (yi2250d) relayed the login information to the Endpoint (salmon).
6. The Endpoint (salmon) logged in to its assigned gateway (yi2250d).

When the Endpoint gave up sending the upcall request and attempted to perform the login to the alternate Endpoint gateway, the following messages appeared in the lcfd.log file. In this log file, the UPCALL_START request (line 5) means the upcall program issued the upcall request and sys=10061 (line 13) means the system is not available.

```
Nov 25 10:00:31 Q lcfd Entering net_wait_for_connection, timeout=-1 handle=0x305ef0
Nov 25 10:03:29 Q lcfd New connection from 127.0.0.1+2022
Nov 25 10:03:29 Q lcfd Entering net_recv, receive a message
Nov 25 10:03:29 Q lcfd Leaving net_recv: bytes=173, (type=16 session=0)
Nov 25 10:03:29 Q lcfd UPCALL_START request
Nov 25 10:03:29 2 lcfd Connecting to '9.3.1.134+9494'
Nov 25 10:03:31 1 lcfd node_login: listener addr '0.0.0.0+9494'
Nov 25 10:03:31 1 lcfd Trying last known gateway ...
Nov 25 10:03:31 Q lcfd login_to_gw
Nov 25 10:03:31 Q lcfd login_gw -> 9.3.1.134+9494
Nov 25 10:03:31 2 lcfd Connecting to '9.3.1.134+9494'
Nov 25 10:03:32 1 lcfd gw login failure: i=2147483647 : ../../src/comm/netio.c:213 [cti_create_client or
cti_timed_create_client] : loc=3, cls=2, dec=7, sys=10061, tli=0, evt=0
Nov 25 10:03:32 Q lcfd Getting next address in login_to_gw...
Nov 25 10:03:32 Q lcfd login_gw -> 9.3.1.134+9494
Nov 25 10:03:32 2 lcfd Connecting to '9.3.1.134+9494'
Nov 25 10:03:34 1 lcfd gw login failure: i=0 : ../../src/comm/netio.c:213 [cti_create_client or
cti_timed_create_client] : loc=3, cls=2, dec=7, sys=10061, tli=0, evt=0
Nov 25 10:03:34 2 lcfd Trying other login listeners...
Nov 25 10:03:34 Q lcfd send_login_dgram: interval=30 attempts=2
Nov 25 10:03:34 Q lcfd net_usend of 570 bytes to 9.3.1.134+9494.  Bcast=0
Nov 25 10:03:34 Q lcfd send_login_dgram: waiting for reply. attempt 1 of 2
Nov 25 10:03:34 Q lcfd net_accept, handle=0x305ef0
Nov 25 10:04:04 Q lcfd send_login_dgram: recv 1 timed out
Nov 25 10:04:04 Q lcfd net_usend of 570 bytes to 9.3.1.134+9494.  Bcast=0
Nov 25 10:04:04 Q lcfd send_login_dgram: waiting for reply. attempt 2 of 2
Nov 25 10:04:04 Q lcfd net_accept, handle=0x305ef0
Nov 25 10:04:34 Q lcfd send_login_dgram: recv 2 timed out
Nov 25 10:04:34 2 lcfd dgram login failure: Timed out
Nov 25 10:04:34 Q lcfd send_login_dgram: interval=30 attempts=2
Nov 25 10:04:34 Q lcfd net_usend of 570 bytes to 9.3.1.149+9494.  Bcast=0
Nov 25 10:04:34 Q lcfd send_login_dgram: waiting for reply. attempt 1 of 2
Nov 25 10:04:34 Q lcfd net_accept, handle=0x305ef0
Nov 25 10:04:39 Q lcfd New connection from 9.3.1.149+2691
Nov 25 10:04:39 Q lcfd Entering net_recv, receive a message
Nov 25 10:04:39 Q lcfd Leaving net_recv: bytes=458, (type=14 session=0)
Nov 25 10:04:39 Q lcfd recv: len='458' (code='14', session='0')
Nov 25 10:04:39 2 lcfd Writing GCS file: C:\Tivoli\lcf\dat\1\last.cfg
Nov 25 10:04:39 1 lcfd salmon is dispatcher 168 in region 1189622596
Nov 25 10:04:39 1 lcfd Logging into new gateway...
Nov 25 10:04:39 Q lcfd login_to_gw
Nov 25 10:04:39 Q lcfd login_gw -> 9.3.1.149+9494
Nov 25 10:04:39 2 lcfd Connecting to '9.3.1.149+9494'
Nov 25 10:04:39 Q lcfd net_send of 442 bytes, session 168
Nov 25 10:04:39 Q lcfd net_accept, handle=0x305ef0
Nov 25 10:04:39 Q lcfd New connection from 9.3.1.149+2692
Nov 25 10:04:39 Q lcfd Entering net_recv, receive a message
Nov 25 10:04:39 Q lcfd Leaving net_recv: bytes=458, (type=14 session=168)
Nov 25 10:04:39 Q lcfd recv: len='458' (code='14', session='168')
Nov 25 10:04:39 2 lcfd Writing GCS file: C:\Tivoli\lcf\dat\1\last.cfg
Nov 25 10:04:39 1 lcfd salmon is dispatcher 168 in region 1189622596
Nov 25 10:04:39 1 lcfd write login file 'lcf.dat' complete
Nov 25 10:04:39 1 lcfd final pid: 113
Nov 25 10:04:39 1 lcfd Login to gateway 9.3.1.149+9494 complete.
```

Case 20:

How is the downcall affected? Figure 68 shows the downcall process flow in this case.

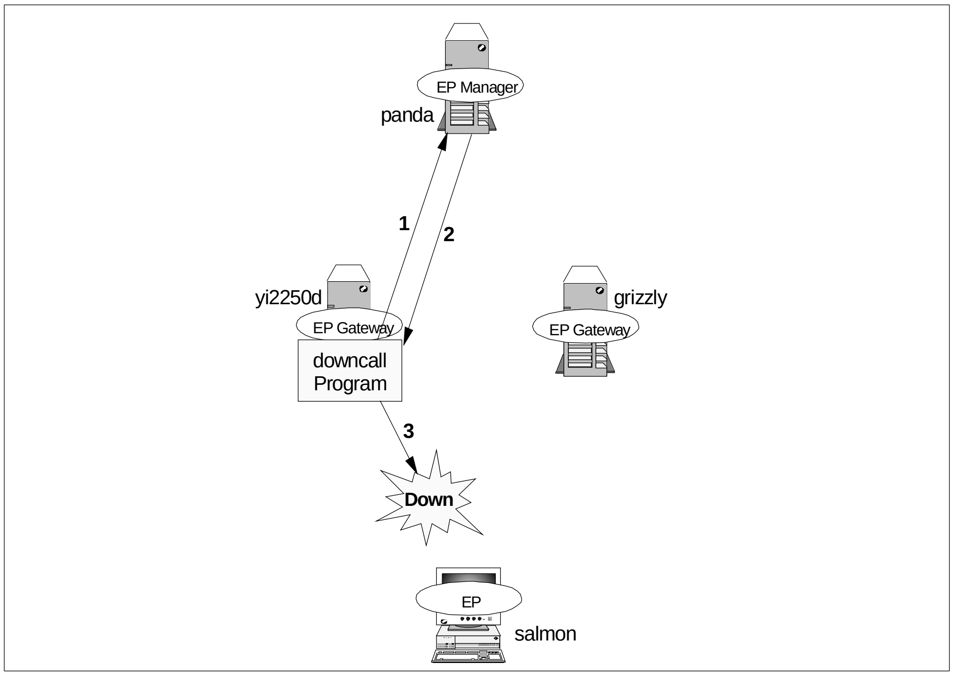


Figure 68. Issuing Downcall when the Endpoint is Unreachable

1. When we invoked the downcall program from the Endpoint gateway (yi2250d), the Endpoint gateway (yi2250d) asked for Endpoint information from the Endpoint Manager (panda).
2. The Endpoint Manager (panda) returned the information to the Endpoint gateway (yi2250d).
3. Then, the Endpoint gateway (yi2250d) attempted to issue the downcall but the Endpoint (salmon) that was the destination of the downcall was unreachable. Therefore the downcall failed.

Normally, our downcall generating program echos back the input characters. The following is a sample output of our downcall program:

```
bash$ dsmain 1189622596.173.554+ hahahaha
Making downcall with hahahaha
[HAHAHAHA]bash$
```

In this case, when we invoked the downcall generating program from the command line on the Endpoint gateway, the following communication error messages were displayed:

```
bash$ dsmain 1189622596.173.554+ hahahaha
Making downcall with hahahaha
DownSamp Exception: A communications failure occurred: IPC shutdown
Please refer to the TME 10 Framework Planning and Installation Guide, "TME Maint
enance and Troubleshooting" for details on diagnosing communication errors or co
ntact your Tivoli support provider.
```

Case 21 and 22:
This case is similar to Cases 19 and 20, but one difference is that this case assumes the Endpoint Manager goes down after the Endpoint login succeeded. After the Endpoint login succeeded, we performed a shutdown of the Endpoint Manager machine. Then we issued the upcall/downcall. We used our sample programs to issue the upcall/downcall as well. We introduce the behavior of the TMA in this situation in the next figure.

Table 32. Case 21 and 22: EP Manager is Unavailable after Login Completion

Case 21 and 22	
Scenario	After the Endpoint login succeeds, we perform a shutdown of the Endpoint Manager. Then we issue the upcall/downcall.

Option	`lcfd -g grizzly+9494`
	`-D bcast_disable=1`

EP Policy	None
Result	***Issuing the upcall:*** The upcall was invoked without problem.
	Issuing the downcall: When the downcall is issued, the EP gateway needs the TMR server for ALI information, so the downcall couldn't be invoked in this case. The downcall failed.

The next figure shows the TMA behavior when we invoked the upcall program.

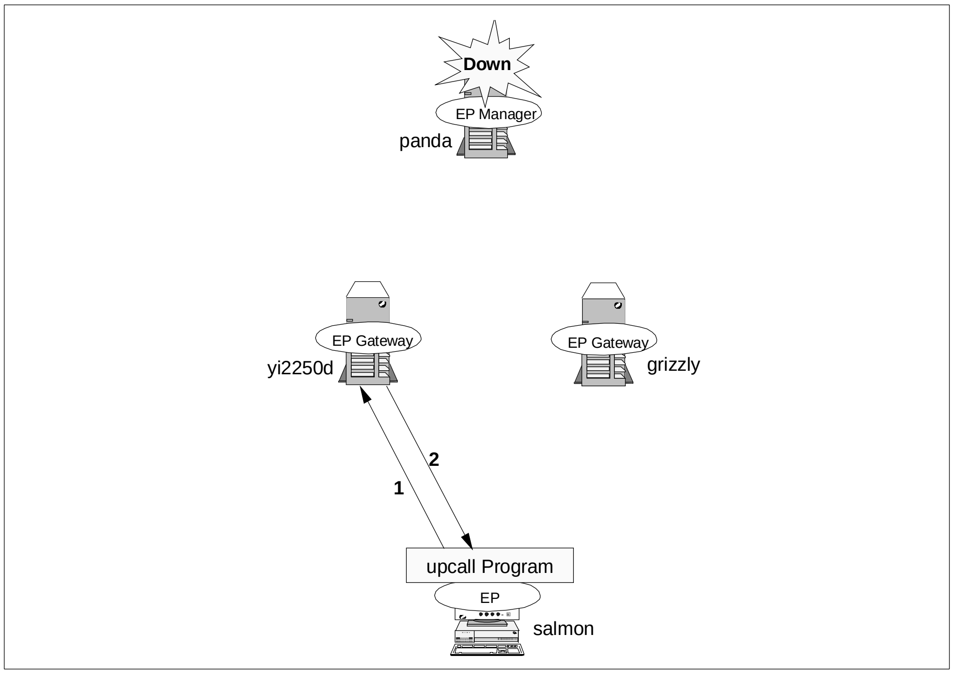


Figure 69. Issuing Upcall when EP Manager is Unavailable

1. Before the testing, the Endpoint (salmon) had logged in to the assigned gateway (yi2250d) successfully. Then we performed a shutdown of the Endpoint Manager (panda). In this situation, we invoked the upcall program to issue the upcall to the assigned gateway (yi2250d).
2. In this case, the Endpoint gateway (yi2250d) performed ALI functions for the Endpoint (salmon), so that the Endpoint gateway (yi2250d) returned the result to the Endpoint (salmon) without making a call to the Endpoint Manager (panda).

Case 22:
The next figure shows the downcall process flow in this case.

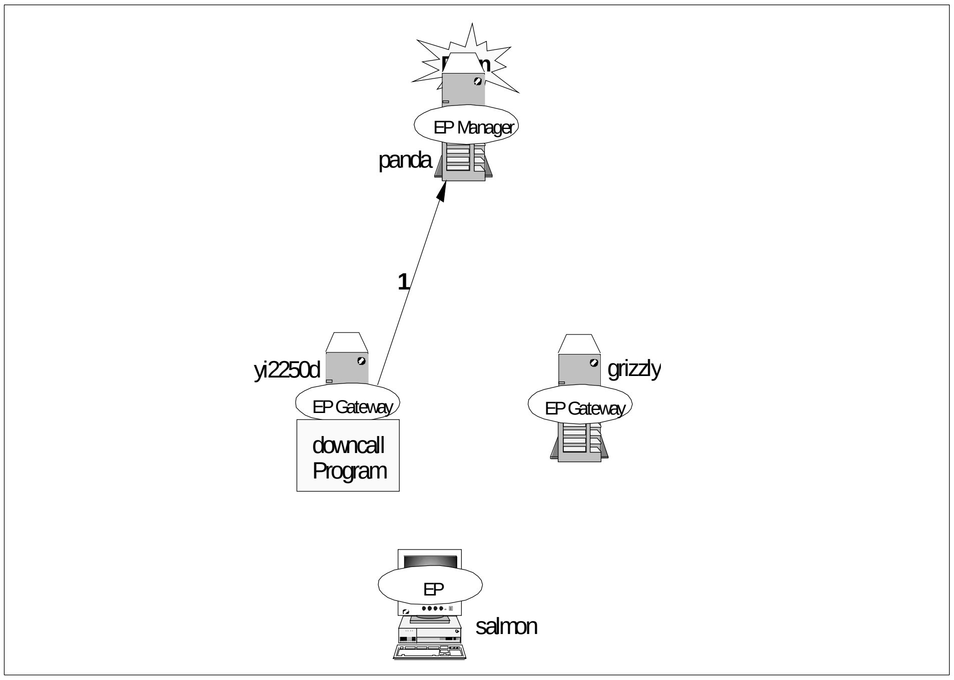


Figure 70. Issuing Downcall when EP Manager is Unavailable

1. As we mentioned in the previous case, when we invoked the downcall program from the Endpoint gateway (yi2250d), the Endpoint gateway (yi2250d) asked for Endpoint information (ALI) from the Endpoint Manager (panda). In this situation, the Endpoint Manager (panda) was not available, so the Endpoint gateway (yi2250d) couldn't get any information from the Endpoint gateway (panda). Therefore the downcall failed.

In this case, we invoked the downcall program from the CLI interface on the Endpoint gateway machine. However, we couldn't get any response, as is indicated in the following:

```
bash$ dsmain 1189622596.173.554+ hahahaha
```

This is a common message in the Tivoli world, especially in situations such as an unavailable DNS server.

5.4.3 Migration

In Chapter 4, “Configuring the TMA Environment” on page 105, we described what migration is and how it works. In this section, we introduce the behavior of the Endpoint Manager, Endpoint gateway and Endpoint while the Endpoint gateway is migrating. We will also talk about the benefit or advantage of migration in Chapter 9, “Management Examples Using TMA” on page 359.

5.4.3.1 Understanding Migration

As we mentioned, we are able to change the gateway assignment made at the initial login using the `wep` command. This is called ’migration’. First of all, we need to understand the function of the `wep` command in our environment and introduce the behavior related to it.

Case 23:

In this case, we attempt to verify what happens when the assigned gateway is migrating.

Table 33. Case 23: EP Gateway Migration

Case 23	
Scenario	Invoke the wep command to perform migration in the multiple EP gateway Environments.
Option	`wep salmon migrate yi2250d`
EP Policy	None
Result	The gateway assignment had been changed as we specified. However, the Endpoint login interface information contained in the lcf.dat file had not yet been modified at the migration.

The following figure shows the processes performed by the `wep` command during migration.

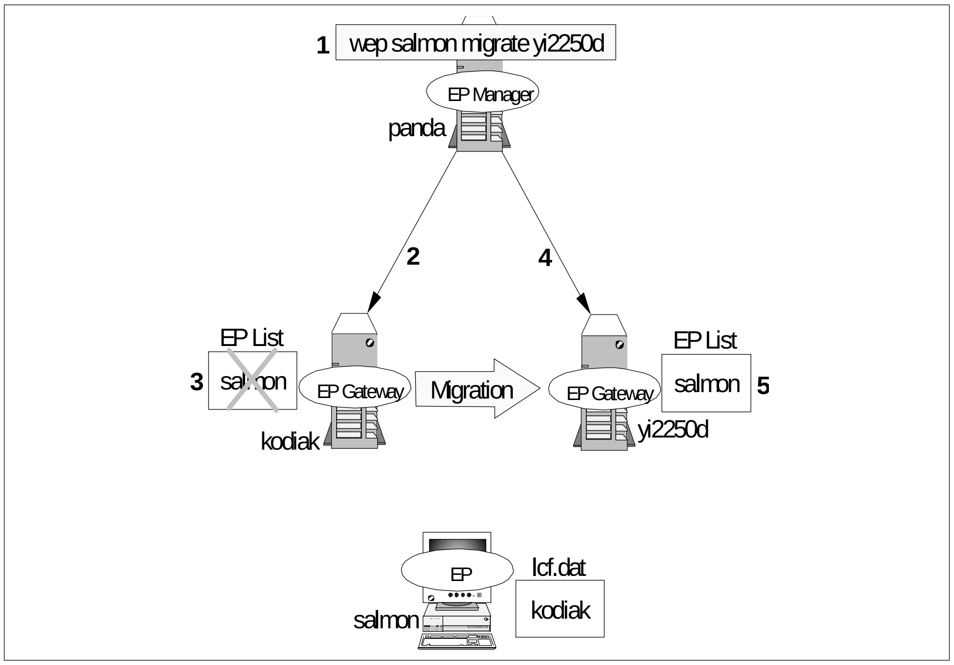


Figure 71. The Endpoint Gateway Migration with the wep Command

1. To perform the migration, we invoked the `wep salmon migrate yi2250d` command from the Endpoint Manager (panda). Then the Endpoint Manager attempted to modify its Endpoint list. As a result, the database file located under the $DBDIR/epmgr.bdb directory was updated.
2. The Endpoint Manager issued the `delete_endpoint` method using the downcall to delete the entry of the migrated Endpoint from the cache in the formerly assigned gateway.
3. The formerly assigned gateway deleted the entry of the Endpoint from its cache.
4. The Endpoint Manager issued the `new_endpoint` method using the downcall to add the entry of the migrated Endpoint into the new assigned gateway's cache as well.
5. The new assigned gateway adds the entry of the Endpoint into the cache.

When you perform the Endpoint gateway migration using the `wep` command, the following messages appear in the epmgrlog file:

```
1998/11/25 17:53:29 +06: get_endpoints: Requesting search for 1189622596.178.508+#TMF_Endpoint::Endpoint#
1998/11/25 17:53:30 +06: migrate 1189622596.178.508+#TMF_Endpoint::Endpoint# -->
1189622596.77.14#TMF_Gateway::Gateway#
1998/11/25 17:53:30 +06: - salmon 1189622596.178.508+#TMF_Endpoint::Endpoint#
1189622596.2.19#TMF_Gateway::Gateway#
1998/11/25 17:53:30 +06: + salmon 1189622596.178.508+#TMF_Endpoint::Endpoint#
1189622596.77.14#TMF_Gateway::Gateway#
1998/11/25 17:53:30 +06: writing epmgr.bdb/1189622596.77.14.bdb for 178
1998/11/25 17:53:30 +06: updating ali map
```

5.4.3.2 Understanding Migration Completion

As you can see from Figure 71 on page 180, there was no communication between the Endpoint gateways and the Endpoint. The Endpoint still had the previous gateway assignment information when the wep command completed. How can the Endpoint recognize the new assigned gateway? In the previous chapter, we described the answer. There are three ways the Endpoint understands what happens and obtains the new gateway assignment information. These ways are called migration completion.

- Completion by the migratory login
- Completion by the migratory upcall
- Completion by the migratory downcall

In this section, we describe the migration completion of each case.

Case 24:
In this case, we attempted to perform the migration completion by the migratory login.

Table 34. Case 24: Migration Completion by EP Login

Case 24	
Scenario	After the migration, to complete the migration by the login, restart the `lcfd` for performing the Endpoint login.

Option	`lcfd -g kodiak+9494`
	`-D bcast_disable=1`

EP Policy	None
Result	The Endpoint attempted to perform the normal login to the formerly assigned gateway, then the Endpoint recognized the migration occurred. After that, the Endpoint logged in to the new assigned gateway.

The following figure shows the detailed processes.

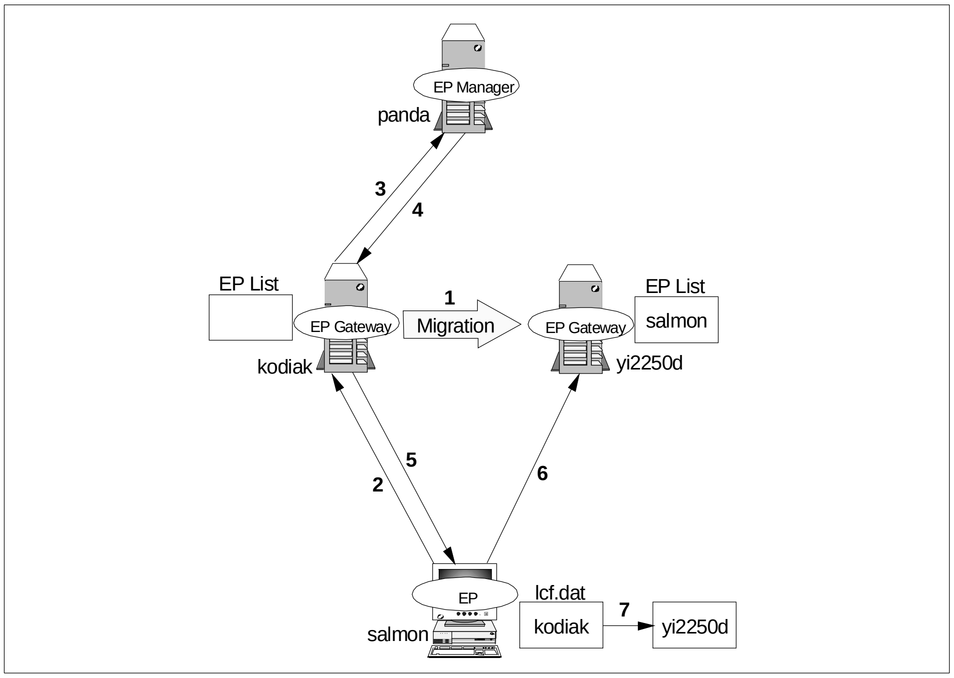


Figure 72. Migration Completion by EP Login

1. We performed the migration with the `wep salmon migrate yi2250d` command.
2. Since the Endpoint still kept the previous gateway assignment information, the Endpoint attempted to perform the normal login to the formerly assigned gateway (kodiak).
3. The formerly assigned gateway (kodiak) received the login request from the Endpoint (salmon), but the formerly assigned gateway (kodiak) no longer manages the Endpoint (salmon), so the Endpoint gateway (kodiak) obtains the new gateway assignment from the Endpoint Manager (panda).
4. The new gateway assignment information was returned to the formerly assigned gateway (kodiak). In this case, of course, the Endpoint gateway selection process didn't occur.
5. The formerly assigned gateway (kodiak) forwarded the new assignment information to the Endpoint (salmon).
6. Then the Endpoint (salmon) recognized the assigned gateway was changed by the migration and obtained the new assignment information.

The Endpoint (salmon) attempted to perform the login to the new assigned gateway (yi2250d).

7. After the Endpoint login completed, the Endpoint (salmon) modified its assigned gateway information stored in the lcf.dat file.

In this case, the formerly assigned gateway recognized the migration had occurred when the formerly assigned gateway received the login request from the Endpoint. Therefore, the following message appears in the gatelog file on the formerly-assigned gateway when the formerly assigned gateway receives the login request from the Endpoint.

```
1998/11/25 18:01:48 +06: eplogin (180): forwarding migration login(3)to epmgr
```

The `lcfd` daemon also writes the following messages to the lcfd.log file when the new assignment information is returned from the formerly assigned gateway and attempts to log in to the new assigned gateway.

```
Nov 25 18:09:49 1 lcfd Trying last known gateway ...
Nov 25 18:09:49 Q lcfd login_to_gw
Nov 25 18:09:49 Q lcfd login_gw -> 9.3.1.133+9494
Nov 25 18:09:49 2 lcfd Connecting to '9.3.1.133+9494'
Nov 25 18:09:49 Q lcfd net_send of 658 bytes, session 180
Nov 25 18:09:49 Q lcfd net_accept, handle=0x306328
Nov 25 18:09:53 Q lcfd New connection from 9.3.1.133+2760
Nov 25 18:09:53 Q lcfd Entering net_recv, receive a message
Nov 25 18:09:53 Q lcfd Leaving net_recv: bytes=674, (type=14 session=0)
Nov 25 18:09:53 Q lcfd recv: len='674' (code='14', session='0')
Nov 25 18:09:53 2 lcfd Writing GCS file: C:\Tivoli\lcf\dat\1\last.cfg
Nov 25 18:09:53 1 lcfd salmon is dispatcher 180 in region 1189622596
Nov 25 18:09:53 1 lcfd endpoint migrated to gateway at 9.3.1.149+9494
Nov 25 18:09:53 1 lcfd Trying last known gateway ...
Nov 25 18:09:53 Q lcfd login_to_gw
Nov 25 18:09:53 Q lcfd login_gw -> 9.3.1.149+9494
Nov 25 18:09:53 2 lcfd Connecting to '9.3.1.149+9494'
Nov 25 18:09:53 Q lcfd net_send of 658 bytes, session 180
Nov 25 18:09:53 Q lcfd net_accept, handle=0x306328
Nov 25 18:09:53 Q lcfd New connection from 9.3.1.149+2988
Nov 25 18:09:53 Q lcfd Entering net_recv, receive a message
Nov 25 18:09:53 Q lcfd Leaving net_recv: bytes=674, (type=14 session=180)
Nov 25 18:09:53 Q lcfd recv: len='674' (code='14', session='180')
Nov 25 18:09:53 2 lcfd Writing GCS file: C:\Tivoli\lcf\dat\1\last.cfg
Nov 25 18:09:53 1 lcfd salmon is dispatcher 180 in region 1189622596
Nov 25 18:09:53 1 lcfd write login file 'lcf.dat' complete
Nov 25 18:09:53 1 lcfd final pid: 105
Nov 25 18:09:53 1 lcfd Login to gateway 9.3.1.149+9494 complete.
```

As you can see, the Endpoint understood the migration had occurred at the line: `endpoint migrated to gateway at 9.3.1.149+9494`.

Case 25:
In this case, we attempted to perform the migration completion by the migratory upcall.

Table 35. Case 25: Migration Completion by Upcall

Case 25	
Scenario	After the migration, to complete the migration by the upcall, execute the upcall program on the Endpoint and the Endpoint issues the upcall to the formerly assigned gateway.

Option	`lcfd -g kodiak+9494`
	`-D bcast_disable=1`

EP Policy	None
Result	The Endpoint issued the upcall to the formerly assigned gateway, then the Endpoint recognized the migration occurred. After that, the Endpoint logged in to the new assigned gateway and issued the upcall again to the new assigned gateway.

The following figure shows the detailed processes.

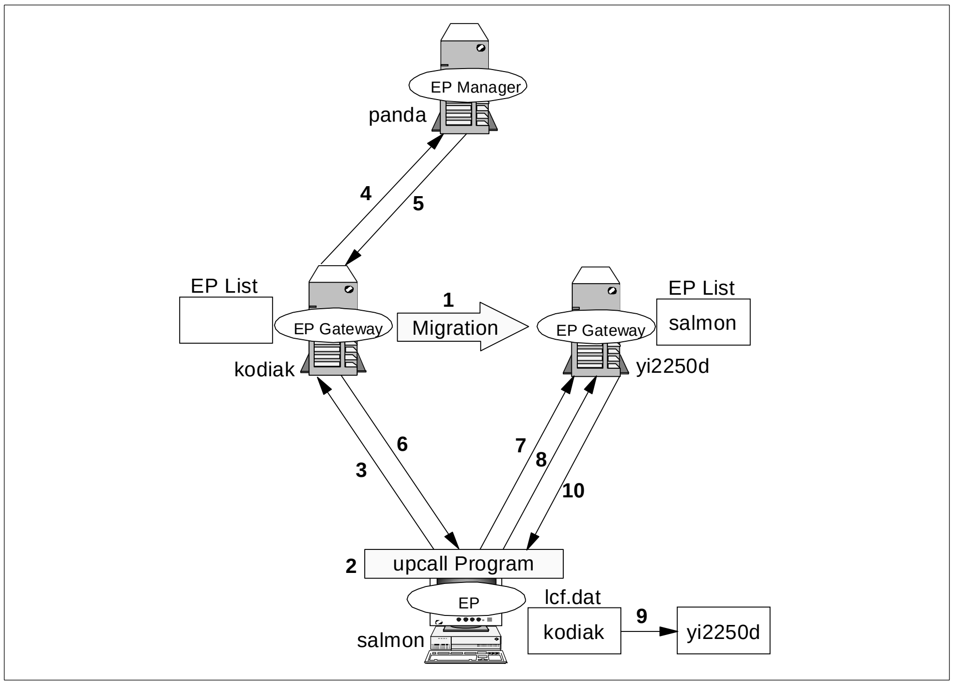


Figure 73. Migration Completion by Upcall

1. We performed the migration with the `wep salmon migrate yi2250d` command.
2. In this situation, we invoked the upcall program to issue the upcall to the Endpoint gateway.
3. Since the Endpoint (salmon) still kept the previous gateway assignment information, the Endpoint attempted to issue the upcall to the formerly assigned gateway (kodiak).
4. The Endpoint gateway (kodiak) which received the upcall no longer managed the Endpoint (salmon), so that the formerly assigned gateway (kodiak) attempted to obtain the new gateway assignment from the Endpoint Manager (panda).
5. The new gateway assignment information returned to the formerly assigned gateway (kodiak). In this case, of course, the Endpoint gateway selection process didn't occur.
6. The formerly assigned gateway (kodiak) forwarded the new assignment information to the Endpoint (salmon).
7. Then the Endpoint (salmon) recognized the assigned gateway was changed by the migration and obtained the new assignment information. The Endpoint (salmon) attempted to perform the login to the new assigned gateway (yi2250d).
8. The Endpoint (salmon) also sent the upcall request to the new assigned gateway (yi2250d).
9. After the Endpoint login completed, the Endpoint (salmon) modified its assigned gateway information stored in the lcf.dat file.
10. Then the Endpoint (salmon) received the result of the upcall from the new assigned gateway (yi2250d).

In this case, the following message appears in the gatelog file on the formerly assigned gateway when the formerly assigned gateway receives the upcall from the Endpoint.

```
1998/11/25 17:52:15 +06: servicing migratory upcall for 178
```

Case 26:

In this case, we attempted to perform the migration completion by the migratory downcall.

Table 36. Case 26: Migration Completion by Downcall

Case 26	
Scenario	After the migration, to complete the migration by the downcall, perform the downcall operation from the EP Manager and the downcall is issued from the new assigned gateway to the Endpoint.

Option	`lcfd -g kodiak+9494`
	`-D dcast_disable=1`

EP Policy	None
Result	The new assigned gateway issued the downcall to the Endpoint, then the Endpoint recognized the migration occurred, so the Endpoint modified its assigned gateway information and returned the result of the downcall to the new assigned gateway.

The following figure shows the detailed processes.

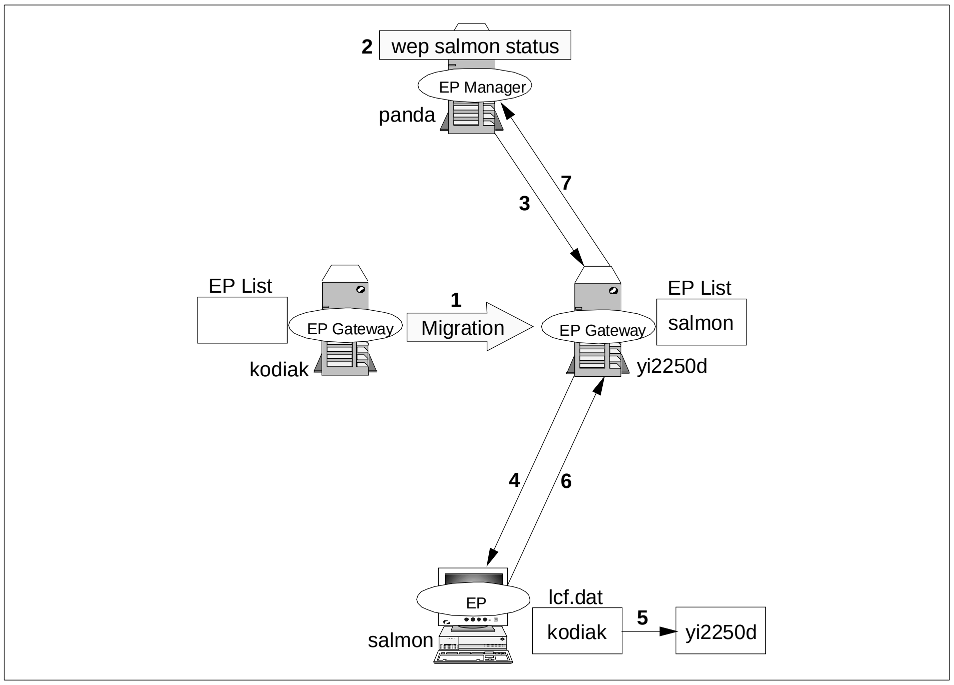


Figure 74. Migration Completion by Downcall

1. We performed the migration with the `wep salmon migrate yi2250d` command.
2. To issue the downcall, we invoked the `wep salmon status` command on the Endpoint Manager (panda).
3. The remote invoke request was sent to the new assigned gateway (yi2250d).
4. The new assigned gateway (yi2250d) issued the downcall to the Endpoint (salmon).
5. Then the Endpoint (salmon) recognized the assigned gateway was changed by the migration and modified its assigned gateway information stored in the lcf.dat file.
6. The Endpoint returned the result of the downcall to the new assigned gateway (yi2250d).
7. The new assigned gateway (yi2250d) also forwarded the result of the downcall to the Endpoint Manager (panda).

When the Endpoint recognizes the assigned gateway is changed by migration and modifies its assigned gateway information, the following messages appear in the lcfd.log file.

```
Nov 25 18:05:40 1 lcfd recording new gateway address 9.3.1.149
Nov 25 18:05:40 1 lcfd write login file 'lcf.dat' complete
```

5.4.3.3 Understanding Migration Completion by Isolation Login

We introduced how the migration processes are completed in the previous sections. These cases assume the formerly assigned gateway is available, but if the formerly assigned gateway is unreachable when the Endpoint is migrating, what happens? In this case, the gateway assignment would depend on the select_gateway_policy if it exists.

Case 27:

In this case, we performed the shutdown of the original assigned gateway, then we invoked the `wep salmon migrate yi2250d` command to perform the migration. At that time, the Endpoint was said to be isolated. We also defined the select_gateway_policy which was inconsistent with the migration. The Endpoint attempted to complete the migration by the isolation login. The most important thing here is which has the priority between the migration or select_gateway_policy. We describe the processes of this case in detail in the next figure.

Table 37. Case 27: Migration Completion by Isolate Login

Case 27	
Scenario	Shutdown the original assigned gateway before the migration and define the select_gateway_policy, which returns a different EP gateway from the new assigned gateway. To complete the migration, the Endpoint attempts to perform the isolate login.

Option	`lcfd -g kodiak+9494`
	`-D dcast_disable=1`

EP Policy	`select_gateway_policy (trout)`
Result	Since the formerly assigned gateway was unreachable, the Endpoint attempted to perform the isolate login to the alternate gateway. During the login process, the EP gateway selection occurred by the select_gateway_policy. As a result, the gateway assignment made at the migration was changed again by the select_gateway_policy. Finally, the Endpoint logged in to the EP gateway which was selected by the select_gateway_policy.

Figure 75 on page 189 shows the detailed processes.

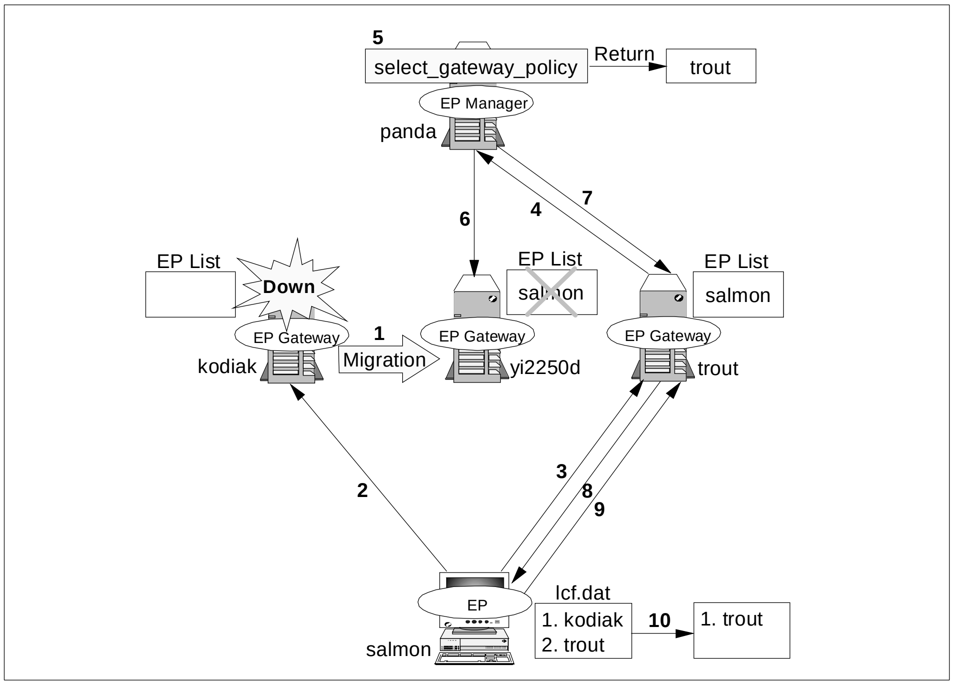


Figure 75. Migration Completion by Isolate Login

1. We performed the migration with the `wep salmon migrate yi2250d` command. At this time, the formerly assigned gateway (kodiak) was unavailable.
2. The Endpoint (salmon) attempted to perform the normal login to the Endpoint gateway (kodiak), however, the Endpoint gateway (kodiak) was not available. The Endpoint (salmon) tried to perform the login until the timeout occurred. In this situation, the Endpoint (salmon) was considered isolated.
3. Then the Endpoint (salmon) attempted to perform the login to the alternate gateway (trout).
4. The alternate gateway forwarded the login request to the Endpoint Manager (panda).
5. The Endpoint Manager (panda) received the login request and referred to the select_gateway_policy. The select_gateway_policy returned the new assigned gateway (trout) information.
6. The new assigned gateway (trout) which the select_gateway_policy returned was different from the assigned gateway (yi2250d) made at the

migration. As a result, the Endpoint Manager (panda) assigned the Endpoint (salmon) to the Endpoint gateway (trout) selected by the policy. The Endpoint Manager (panda) modified its Endpoint list and also updated the cache on the assigned gateway (yi2250) made at the migration.

7. The Endpoint Manager (panda) returned the gateway assignment information to the Endpoint gateway (trout).
8. The Endpoint gateway (trout) updated its cache and relayed the assignment information.
9. Finally, the Endpoint (salmon) logged in to the gateway (trout) assigned by the select_gateway_policy.
10. The Endpoint (salmon) modified its assigned gateway information stored in the lcf.dat file.

Note

When you perform the migration from Endpoint gateway A to Endpoint g gatewayateway B, if Endpoint gateway A is not available, what happens? You can migrate even if the source gateway is unavailable. When Endpoint gateway A boots, the Endpoint gateway attempts to synchronize its cache (gwdb.bdb file) and then the Endpoint gateway recognizes the migration has been performed.

In this case, the gateway assignment was changed twice, so the following messages appeared in the epmgrlog file:

```
1998/11/25 18:33:11 +06: get_endpoints: Requesting search for 1189622596.184.508+#TMF_Endpoint::Endpoint#
1998/11/25 18:33:11 +06: migrate 1189622596.184.508+#TMF_Endpoint::Endpoint# -->
1189622596.77.14#TMF_Gateway::Gateway#
1998/11/25 18:33:11 +06: - salmon 1189622596.184.508+#TMF_Endpoint::Endpoint# 1189622596.2.19
1998/11/25 18:33:11 +06: + salmon 1189622596.184.508+#TMF_Endpoint::Endpoint#
1189622596.77.14#TMF_Gateway::Gateway#
1998/11/25 18:33:11 +06: writing epmgr.bdb/1189622596.77.14.bdb for 184
1998/11/25 18:33:12 +06: updating ali map
1998/11/25 18:33:14 +06: iom query 1189622596.4.21
1998/11/25 18:33:14 +06: iom query 1189622596.2.19
1998/11/25 18:33:14 +06: iom query 1189622596.109.19
1998/11/25 18:33:14 +06: iom query 1189622596.77.14
1998/11/25 18:33:43 +06: dispatcher 184 logging in with code 2
1998/11/25 18:33:46 +06: gw::new_endpoint(1189622596.2.19) :
1998/11/25 18:33:46 +06: 1189622596.184.508+ assigned to 1189622596.109.19
1998/11/25 18:33:46 +06: - salmon 1189622596.184.508+#TMF_Endpoint::Endpoint#
1189622596.77.14#TMF_Gateway::Gateway#
1998/11/25 18:33:46 +06: + salmon 1189622596.184.508+#TMF_Endpoint::Endpoint# 1189622596.109.19
1998/11/25 18:33:46 +06: writing epmgr.bdb/1189622596.109.19.bdb for 184
1998/11/25 18:33:47 +06: updating ali map
```

As you can see, in this case, the Endpoint didn’t recognize the migration occurred. For the Endpoint, this case was the same as the isolated login. The assigned gateway also didn’t recognize that the migration occurred in this case, so the following messages appeared in the gatelog file on the formally assigned gateway when it received the login request from the Endpoint.

```
1998/11/25 18:41:32 +06: eplogin (0): forwarding isolation login(2)to epmgr
```

Note

In this case, we defined the select_gateway_policy, but if the Endpoint policy didn't exist, what happens? Of course, the Endpoint would log in to the newly assigned gateway (yi2250d) and the alternate gateway (trout) will be the intercepting gateway.

5.5 TMA 3.2 and TMA 3.6

Version 3.2 of Tivoli supported the LCF architecture for the first time and implemented the `lcfd` daemon as part of the Tivoli Management Framework services. Now, Version 3.6 of Tivoli is available, so different versions of the TMA (`lcfd`) might exist in a single TMR environment. Since the TMA has an auto upgrade feature, there is nothing to be concerned about. In this section, we introduce the auto upgrade of the TMA (`lcfd`) and also verify the auto upgrade process.

5.5.1 What is Auto Upgrade?

For most of customers, version upgrade issues are a serious problem, especially in a large environment. This isn't a problem for the TMA and the Tivoli Management Applications running on the TMA.

The Endpoint gateways have an upgrade package for upgrading the Endpoint software under the $BINDIR/../lcf_bundle directory. If the Endpoint that attempts to log in to the Endpoint gateway is a lower version of the `lcfd`, the Endpoint gateway performs an upgrade of the Endpoint software using the upgrade package the Endpoint gateway stores. This is the auto upgrade of the TMA.

The auto upgrade is a very clever implementation in the LCF architecture. The following is an overview of the auto upgrade process.

1. The `lcfd` attempts to perform the login to the appropriate Endpoint gateway.

2. The Endpoint gateway accepts the login request from the Endpoint and processes it. The Endpoint gateway also checks the version of the Endpoint during the login process.
3. If the Endpoint gateway detects a lower version of the Endpoint, the Endpoint gateway invokes the `wadminep` command to perform the upgrade of the Endpoint (`lcfd`) automatically.

5.5.2 Auto Upgrade of TMA

To enable the auto upgrade the TMA, we need to perform a few simple configuration steps. In this section we introduce how to enable auto upgrade.

5.5.2.1 How to Enable an Auto Upgrade

We already talked about the Endpoint policy in Chapter 4, “Configuring the TMA Environment” on page 105. The auto upgrade uses the Endpoint login_policy executed in the Endpoint gateway at every Endpoint login. The following two steps enable auto upgrade:

1. Define login_policy, including the upgrade.sh script.
2. Change mode in the upgrade.cntl file from disabled to auto.

Defining Endpoint login_policy:
First, you have to define the Endpoint login_policy including the auto upgrade logic (`upgrade.sh`) to enable the auto upgrade (refer to the sample login_policy shown in *The Framework Reference Manual*). Put the following line into the login_policy:

```
$BINDIR/../lcf_bundle/upgrade/upgrade.sh $1 $8 $3
```

As we mentioned before, the Endpoint login_policy is invoked on the Endpoint gateway every time the Endpoint attempts to perform the login to the gateway even if it is a normal login. After editing the login_policy for putting the `upgrade.sh` into the policy, the auto upgrade process should be invoked every time the Endpoint logs in. As a result, the upgrade.sh would check the `upgrade_mode` defined in the upgrade.cntl file, then, if it is configured as auto mode, the `upgrade.sh` will upgrade the Endpoint software if the available version in the EPUPG.INF file is greater than the current version of the Endpoint.

Note

The upgrade.sh script uses the EPUPG.INF file located under the $BINDIR/../lcf_bundle/bin/<interps>/upgrade directory to decide whether the available version is greater than the current version of the Endpoint or not. The following are the contents of the EPUPG.INF file (for Windows NT) and the version information of this file is used for auto upgrade.

```
file(bin)=lcfd.exe
file(bin)=lcfep.exe
file(lib)=libmrt.dll
file(lib)=libcpl.dll
file(lib)=libdes.dll
file(lib)=libccms_lcf.dll
file(endpoint->bin)=wlcftap.exe
file(endpoint->bin)=ntconfig.exe
file(endpoint->bin)=libacct.dll
file(endpoint->system)=TivoliAP.dll
file(endpoint->bin)=MSVCRT40.DLL
file(dat)=lcf_env.sh
file(dat)=lcf_env.cmd
file(inf)=EPUPG.INF
current_version=5
previous_version=2.1
```

Changing Mode of Auto Upgrade Control File:
The Endpoint gateways have a control file for enabling the auto upgrade and the control file is located at:

```
/usr/local/Tivoli/bin/lcf_bundle/upgrade/upgrade.cntl
```

The upgrade.cntl file is a quick way to turn on or off the auto upgrade function. To enable the auto upgrade function, edit the upgrade.cntl file as follows. By default, upgrade_mode is disabled.

```
# AUTO UPGRADE CONTROL FILE
# To enable auto upgrade for this gateway, set the mode to 'auto'.
#
#HOW DOES IT WORK?
#The login_policy.sh script is executed each time an endpoint
#logs into the Gateway
#This script executes the upgrade.sh script which
#checks the current version of the endpoint against the available
#version in the lcf_bundle.  If upgrade is needed, the upgrade.sh
#script
#executes wadminep in upgrade mode.
#
upgrade_mode=auto
```

The upgrade.sh greps the upgrade.cntl file for the word auto and exits if it is not found. After editing it, the Endpoint gateway performs the auto upgrade process if it is a lower version during the Endpoint initial login process. The following figure (Figure 76 on page 194) shows the flow of these processes.

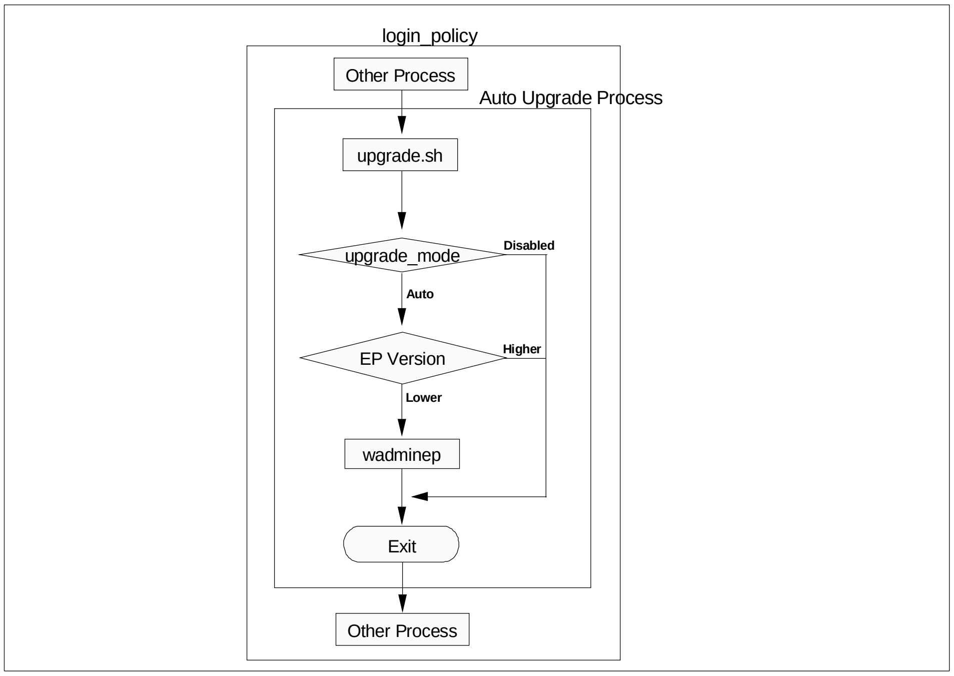


Figure 76. The Process Flow of the Auto Upgrade Function

This is a very useful function. We can define only one login_policy in a single TMR. This means we cannot control the auto upgrade function for each Endpoint gateway using the login_policy. However, the upgrade.cntl file allows us to control turning on or off the auto upgrade for each Endpoint gateway.

Note

If you don't configure anything for auto upgrade, nothing will happen. Therefore, you have to configure for auto upgrade using the preceding method if you would like to use this feature. However, we don't recommend that you enable the auto upgrade unless you know that an upgrade is available, because the auto upgrade may affect the performance of the Endpoint login, especially in very large environment. The upgrade.sh uses the `awk` or `grep` commands, so the impact on the system performance may be significant. If a hundred Endpoints attempt to perform the login and the upgrade.sh script runs for each Endpoint, what will happen? The login_policy will be executed for every Endpoint login. We will talk about this in more detail in Chapter 10, "Tivoli Management Agent Performance Conisderations" on page 385.

5.5.2.2 Using Auto Upgrade of TMA

When you configure the auto upgrade and the auto upgrade function achieves the upgrading process of the TMA, the following messages would be written in the lcfd.log file. In this case, we performed the upgrade of the TMA from Version 2.1 to Version 5.

```
Dec 18 16:57:36 1 lcfd Spawning:
C:\Tivoli\lcf\dat\12181657.228\cache\bin\w32-ix86\endpoint\admin.exe, ses: 04758f29
Dec 18 16:57:36 1 admin Mode = view_interpreter
Dec 18 16:57:38 1 lcfd Spawning:
C:\Tivoli\lcf\dat\12181657.228\cache\bin\w32-ix86\endpoint\admin.exe, ses: 04758f2a
Dec 18 16:57:39 1 admin_rpt extract_header, key = 5
Dec 18 16:57:39 1 admin_rpt extract_header, key = 2.1
Dec 18 16:57:39 1 admin_rpt extract_header, key = lcfd.exe
Dec 18 16:57:39 1 admin_rpt extract_header, key = 2
Dec 18 16:57:39 1 admin_rpt extract_header, key = 90112
Dec 18 16:57:39 1 admin_rpt extract_header, key = lcfep.exe
Dec 18 16:57:39 1 admin_rpt extract_header, key = 2
Dec 18 16:57:39 1 admin_rpt extract_header, key = 108544
Dec 18 16:57:39 1 admin_rpt extract_header, key = libmrt.dll
Dec 18 16:57:39 1 admin_rpt extract_header, key = 1
Dec 18 16:57:39 1 admin_rpt extract_header, key = 210432
Dec 18 16:57:39 1 admin_rpt extract_header, key = libcpl.dll
Dec 18 16:57:39 1 admin_rpt extract_header, key = 1
Dec 18 16:57:39 1 admin_rpt extract_header, key = 15872
Dec 18 16:57:39 1 admin_rpt extract_header, key = libdes.dll
Dec 18 16:57:39 1 admin_rpt extract_header, key = 1
Dec 18 16:57:39 1 admin_rpt extract_header, key = 17920
Dec 18 16:57:39 1 admin_rpt extract_header, key = libccms_lcf.dll
Dec 18 16:57:39 1 admin_rpt extract_header, key = 1
Dec 18 16:57:39 1 admin_rpt extract_header, key = 46592
Dec 18 16:57:39 1 admin_rpt extract_header, key = wlcftap.exe
Dec 18 16:57:39 1 admin_rpt extract_header, key = 0
Dec 18 16:57:39 1 admin_rpt extract_header, key = 29696
Dec 18 16:57:39 1 admin_rpt extract_header, key = ntconfig.exe
Dec 18 16:57:39 1 admin_rpt extract_header, key = 0
Dec 18 16:57:39 1 admin_rpt extract_header, key = 5632
Dec 18 16:57:39 1 admin_rpt extract_header, key = libacct.dll
Dec 18 16:57:39 1 admin_rpt extract_header, key = 0
Dec 18 16:57:39 1 admin_rpt extract_header, key = 34816
Dec 18 16:57:39 1 admin_rpt extract_header, key = TivoliAP.dll
Dec 18 16:57:39 1 admin_rpt extract_header, key = 0
Dec 18 16:57:39 1 admin_rpt extract_header, key = 32256
Dec 18 16:57:39 1 admin_rpt extract_header, key = MSVCRT40.DLL
Dec 18 16:57:39 1 admin_rpt extract_header, key = 0
Dec 18 16:57:39 1 admin_rpt extract_header, key = 312832
Dec 18 16:57:39 1 admin_rpt extract_header, key = lcf_env.sh
Dec 18 16:57:39 1 admin_rpt extract_header, key = 0
Dec 18 16:57:39 1 admin_rpt extract_header, key = 1690
Dec 18 16:57:39 1 admin_rpt extract_header, key = lcf_env.cmd
Dec 18 16:57:39 1 admin_rpt extract_header, key = 0
Dec 18 16:57:39 1 admin_rpt extract_header, key = 667
Dec 18 16:57:39 1 admin_rpt extract_header, key = EPUPG.INF
Dec 18 16:57:39 1 admin_rpt extract_header, key = 0
Dec 18 16:57:39 1 admin_rpt extract_header, key = 396
Dec 18 16:57:39 1 admin_rpt extract_header, version = 5, prev_version = 2.1
Dec 18 16:57:39 1 admin_rpt System directory has been determined as 'C:\WINNT'
Dec 18 16:57:41 1 admin_rpt file = C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\lcfd.exe,
type = 2, size = 90112
Dec 18 16:57:42 1 admin_rpt file = C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\lcfep.exe,
type = 2, size = 108544
Dec 18 16:57:45 1 admin_rpt file = C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\libmrt.dll,
type = 1, size = 210432
Dec 18 16:57:47 1 admin_rpt file = C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\libcpl.dll,
type = 1, size = 15872
Dec 18 16:57:47 1 admin_rpt file = C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\libdes.dll,
type = 1, size = 17920
Dec 18 16:57:48 1 admin_rpt file =
C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\libccms_lcf.dll, type = 1, size = 46592
Dec 18 16:57:48 1 admin_rpt file =
C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\wlcftap.exe, type = 0, size = 29696
Dec 18 16:57:50 1 admin_rpt file =
C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\ntconfig.exe, type = 0, size = 5632
Dec 18 16:57:50 1 admin_rpt file =
C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\libacct.dll, type = 0, size = 34816
Dec 18 16:57:51 1 admin_rpt file =
C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\TivoliAP.dll, type = 0, size = 32256
Dec 18 16:57:51 1 admin_rpt file =
C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\MSVCRT40.DLL, type = 0, size = 312832
Dec 18 16:57:56 1 admin_rpt file = C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\lcf_env.sh,
type = 0, size = 1690
Dec 18 16:57:56 1 admin_rpt file =
C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\lcf_env.cmd, type = 0, size = 667
Dec 18 16:57:56 1 admin_rpt file = C:\Tivoli\lcf\bin\w32-ix86\mrt\upgrade\EPUPG.INF,
type = 0, size = 396
Dec 18 16:57:56 1 admin_rpt Removing service = lcfd
Dec 18 16:57:56 1 admin_rpt shutdown_lcfd, process =
C:\Tivoli\lcf\bin\w32-ix86\mrt\lcfd.exe
Dec 18 16:58:06 1 admin_rpt Starting service
Dec 18 16:58:06 1 admin_rpt startup_lcfd, process =
C:\Tivoli\lcf\bin\w32-ix86\mrt\lcfd.exe
```

5.6 Co-Existing Endpoints and EP Gateways on the Same Node

Sometimes, we may need to configure an Endpoint and Endpoint gateway on a single machine. The Endpoint and Endpoint gateway can work together on a single machine. However, in this case, there are some considerations for implemention.

By default, the port number that the Endpoint gateway uses and the Endpoint uses is the same, port 9494. If the Endpoint gateway and Endpoint attempt to use the same port on a single machine, only the first process is available and the second will be unavailable because of a port busy error.

Normally, on the UNIX platform, the `lcfd` daemon starts first, so that the Endpoint gateway process (`gateway`) becomes unavailable. Then the following message appears in the gatelog file:

```
net_create_remote_server: port 9494 is already in use. Waiting...
gateway boot FAILED: Can't bind to 0.0.0.0+9494: address is already in use.
```

If the `lcfd` daemon cannot open the port 9494, the `lcfd` daemon writes the login failure message into the lcfd.log file.

To avoid this situation, you need to configure a different port number for the Endpoint gateway and the Endpoint which are running on the same machine. If you change the default port number of the Endpoint gateway, this will affect all Endpoints which have already logged into the Endpoint gateway. Therefore, we strongly recommend that you change the port number of the Endpoint. To change the port number of the Endpoint, you can use the following methods:

- The setup.iss file at the Endpoint installation.
- The last.cfg file.
- The `lcfd` daemon option (-P or -D lcfd_port)

Note

With newer versions of the Tivoli Management Framework (Version 3.6.1), the port number 9495 is the default port number for the Endpoint. This means that an Endpoint can be configured on an Endpoint gateway machine without a conflict.

5.7 Problem Determination

In this chapter, we have introduced various aspects of Endpoint behavior. To implement the TMA in a real environment, it is very important to understand the TMA behavior. However, sometimes you may have a problem which you have never seen. This section provides an overview of TMA problem determination and debugging information for resolving problems.

5.7.1 Generic Problem Determination

If an Endpoint could not log into any Endpoint gateways, the points to check are:

- Did you create the Endpoint gateway?
- Is the Endpoint gateway up and reachable?
- Is it listening on the port expected by the `lcfd` daemon?

Use the `wgateway` command to confirm whether the Endpoint gateway is reachable and which port it is listening on.

```
C:\>wgateway
Object                   Name                 Status
1189622596.4.21          grizzly-gateway      u
1189622596.2.19          kodiak-gateway       d
1189622596.195.21        yi2250d-gateway      u
C:\>
```

To see detailed information, use the `gw_label` command as follows:

```
C:\>wgateway kodiak-gateway
Object      : 1189622596.2.19#TMF_Gateway::Gateway#
Hostname    : kodiak
Port        : 9494
Timeout     : 300
Debug level : 9
C:\>
```

Use the following command to start the Endpoint gateway:

```
wgateway kodiak-gateway start
```

If you cannot detect any problem through the above operations, you should check the log files regarding the TMA. The next part of this section introduces the log files for TMA problem determination.

5.7.2 Debugging Information

In this section, we classify the debugging information provided by the Endpoint Manager, Endpoint gateway and Endpoint for making problem determination easier. This information also helps us to decide which debug (log) level is the best for normal operation, or tells us what kind of debug information we can get from the log files, and so on.

5.7.2.1 Log File Location

In the TMA environment, each of the components, the Endpoint Manager, Endpoint gateway and Endpoint, have a log file shown in Figure 77 on page 199. These log files help us to trace TMA behavior.

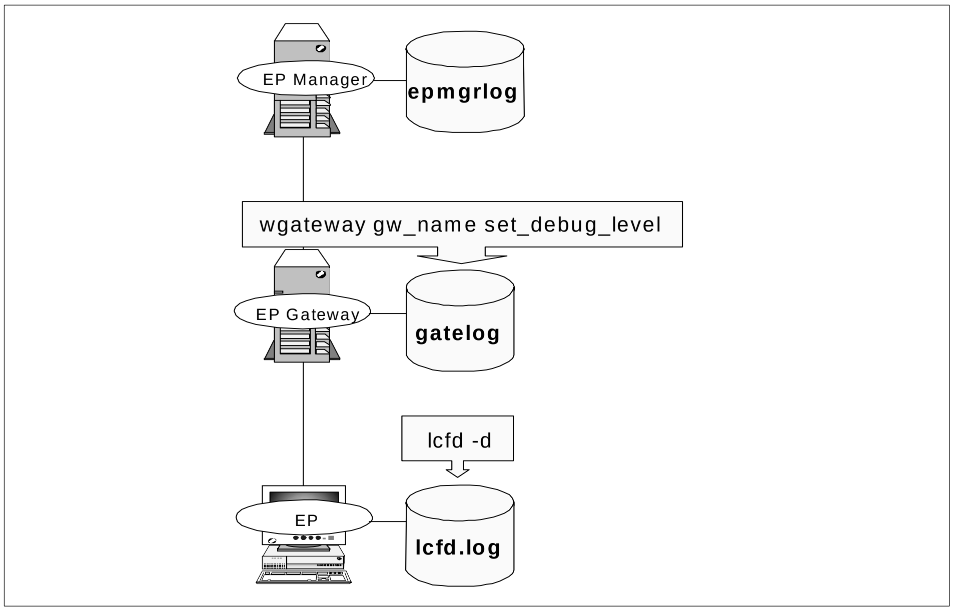


Figure 77. The Location of the TMA Log Files

5.7.2.2 epmgrlog File

The epmgrlog file is located in the $DBDIR/epmgrlog (or %DBDIR%\epmgrlog for NT) directory of the Endpoint Manager machine. This log file contains information related to Endpoint Manager behavior, so you can check what sort of processes have taken place on the Endpoint Manager. In Version 3.6 of the Tivoli Management Framework, the debug level of the epmgrlog file is fixed, so we are not able to change the debug level. The following sample shows the messages logged in the epmgrlog file when the Endpoint performs the initial login.

```
1998/11/23 15:14:28 +06: 1189622596.126.508+#TMF_Endpoint::Endpoint# assigned to
1189622596.2.19#TMF_Gateway::Gateway#
1998/11/23 15:14:29 +06: + salmon 1189622596.126.508+#TMF_Endpoint::Endpoint#
1189622596.2.19#TMF_Gateway::Gateway#
1998/11/23 15:14:29 +06: writing epmgr.bdb/1189622596.2.19.bdb for 126
1998/11/23 15:14:30 +06: updating ali map
1998/11/23 15:14:33 +06: update 1189622596.126.508+#TMF_Endpoint::Endpoint#
1998/11/23 15:14:33 +06: rewriting epmgr.bdb/1189622596.2.19.bdb for 126
```

Normally, most of the Endpoint troubles occur at the Endpoint login. For example, the Endpoint cannot log into any Endpoint gateway. The messages in the epmgrlog file are logged after the Endpoint gateway receives the Endpoint login request, so we may not see useful information in the epmgrlog file if the problem exists between the Endpoint and the gateway.

5.7.2.3 gatelog File

The gatelog file is located in the $DBDIR/gatelog (or %DBDIR%\gatelog for NT) directory of the Endpoint gateway machine. This log file contains the information related to Endpoint gateway behavior and we can change the debug level from 0 to 9 using the `wgateway` command. The following are contents for each debug level of the gatelog file.

Table 38. The Contents of the gatelog File

Debug Level	Description
0	Only errors and likely errors are logged. This is the default and recommended debug level.
1	Things that are exceptional, that may or may not be errors are logged (warnings).
2	Things that are exceptional, but we aren’t concerned about them at this time.
3	Verbose communication information, since that's where most errors are likely to be diagnosed.
4	This debug level doesn’t exist.
5	Verbose boot, database check of the gateway, and Endpoint login information.
6	Verbose upcall, downcall, and repeater information.
7	Verbose job scheduler information.
8	Verbose gateway cache information.
9	Things that only a gateway developer would want to see.

If you attempt to check the log (gatelog) file of the Endpoint gateway, the Endpoint gateway should be configured at debug level 9 because the information provided by the default debug level (=0) is not enough to detect problems. Use the following command to do this.

```
wgateway kodiak-gateway set_debug_level 9
```

This takes effect immediately without restarting the Endpoint gateway. You can check the gatelog file ($DBDIR/gatelog) to see what sort of processes have taken place on the Endpoint gateway. The Endpoint initial login sequence is logged in the gatelog file as follows.

```
1998/11/23 15:12:04 +06: sched: got a job
1998/11/23 15:12:04 +06: process_node_login: Endpoint is speaking ECP protocol version 2
1998/11/23 15:12:04 +06: processing login request from 9.3.1.193+1140
(salmon,w32-ix86,G0WCGGLYS6S28XKJ3XKR0000059F,reg=0,od=0)
1998/11/23 15:12:04 +06: eplogin (0): forwarding initial login to epmgr
1998/11/23 15:12:13 +06: login succeeded for 9.3.1.193+1140
(salmon,w32-ix86,G0WCGGLYS6S28XKJ3XKR0000059F,reg=1189622596,od=126)
1998/11/23 15:12:14 +06: reconnect_thread: connection from 9.3.1.193+1141
1998/11/23 15:12:14 +06: tcp server: waiting for connection on 0.0.0.0+9494...
1998/11/23 15:12:14 +06: reader_thread: received data: session=7e, type=13, len=722
1998/11/23 15:12:14 +06: sched: got a job
1998/11/23 15:12:14 +06: process_node_login: Endpoint is speaking ECP protocol version 2
1998/11/23 15:12:14 +06: processing login request from 9.3.1.193+1141
(salmon,w32-ix86,G0WCGGLYS6S28XKJ3XKR0000059F,reg=1189622596,od=126)
1998/11/23 15:12:14 +06: we need to send codeset 1252
1998/11/23 15:12:14 +06: process_node_login: epcache hit
1998/11/23 15:12:14 +06: process_node_login: Updating epcache with new encryption type.
1998/11/23 15:12:14 +06: process_node_login: Updating epcache with new ECP version.
1998/11/23 15:12:14 +06: login succeeded for 9.3.1.193+1141
(salmon,w32-ix86,G0WCGGLYS6S28XKJ3XKR0000059F,reg=1189622596,od=126)
1998/11/23 15:12:14 +06: new_session: 1ea042ff, connecting to 9.3.1.193+9494...
1998/11/23 15:12:14 +06: destroying session 1ea042ff
1998/11/23 15:12:15 +06: gwcache: miss key=<1189622596.1.508,.meth.,write_html>
1998/11/23 15:12:15 +06: gwcache: hit key=<1189622596.1.508,.inh.,>
1998/11/23 15:12:15 +06: gwcache: miss key=<1189622596.1.510,.meth.,write_html>
1998/11/23 15:12:15 +06: gwcache: hit key=<1189622596.1.510,.inh.,>
1998/11/23 15:12:15 +06: gwcache: hit key=<1189622596.1.509,.meth.,write_html>
1998/11/23 15:12:15 +06: gwcache: hit key=<1189622596.1.516#Depends::Mgr#,.attr.,_info>
1998/11/23 15:12:15 +06: downcall: Method body /bin/w32-ix86/endpoint/msg_bind found.
1998/11/23 15:12:15 +06: downcall: dependency /msg_cat/C/GatewayCatalog.cat found.
1998/11/23 15:12:15 +06: downcall: dependency /msg_cat/fr_FR/GatewayCatalog.cat found.
1998/11/23 15:12:15 +06: downcall: dependency /msg_cat/ja_JP/GatewayCatalog.cat found.
1998/11/23 15:12:15 +06: downcall: dependency /msg_cat/pt_BR/GatewayCatalog.cat found.
1998/11/23 15:12:15 +06: idmap: user ($root_user,w32-ix86) -> Administrator
1998/11/23 15:12:15 +06: new_session: 1ea04300, connecting to 9.3.1.193+9494...
1998/11/23 15:12:16 +06: reader_thread: received data: session=1ea04300, type=5, len=60
1998/11/23 15:12:16 +06: destroying session 1ea04300
1998/11/23 15:12:16 +06: run_login_policy: Running login policy on endpoint salmon.
1998/11/23 15:12:19 +06: run_ep_boot_methods: nothing to do.
```

5.7.2.4 lcfd.log File

The lcfd.log file is normally located in the following directory on the Endpoint:

NT	C:\Tivoli\lcf\dat\1
UNIX	/opt/Tivoli/lcf/dat/1

The contents of the lcfd.log file can be changed by specifying the message level using the -d or -D log_threshold option with the `lcfd` daemon. The following table shows the five levels of messages generated by Endpoints.

Table 39. The Message Contents of the lcfd.log File

Message Level	Message Content
0	No message logging.

Message Level	Message Content
1	Minimal logging. This is the default and recommended message level.
2	Tracing and moderate output.
3	Detailed information and tight loops.
4	Data buffers. It is too detailed for us to analyze. This is only for development consideration.

The log file (lcfd.log) of the Endpoint (`lcfd` daemon) is the most useful when determining the problems regarding Endpoint behavior. We recommend that you configure the `log_threshold` at level 2 to determine the problem because `log_threshold`=3 or 4 provides too much detailed information. You can configure the `log_threshold` option using the `-d 2` option of the `lcfd` daemon during the Endpoint installation, or by using the Endpoint Web interface.

In the Endpoint log file, you can check the address of the Endpoint gateway with which the Endpoint is trying to communicate. The Endpoint initial login sequence is logged in the lcfd.log file as follows (where `log_threshold`=2).

```
Performing browse mode ’view_file’ on endpoint ’salmon’
Nov 23 15:20:19 Q lcfd Entering CacheInit
Nov 23 15:20:19 1 lcfd CacheInit: Starting new index file:
C:\Tivoli\lcf\dat\1\cache\Index.v5
Nov 23 15:20:19 Q lcfd CacheSetPurgeMethod: CACHE_PURGE_OLDEST
Nov 23 15:20:19 2 lcfd Writing GCS file: C:\Tivoli\lcf\dat\1\last.cfg
Nov 23 15:20:19 1 lcfd lcfd 5 (w32-ix86)
Nov 23 15:20:19 1 lcfd Binary Dir (load_dir): ’C:\Tivoli\lcf\bin\w32-ix86\mrt’
Nov 23 15:20:19 1 lcfd Library Dir (lib_dir): ’C:\Tivoli\lcf\bin\w32-ix86\mrt’
Nov 23 15:20:19 1 lcfd Dat Dir (run_dir): ’C:\Tivoli\lcf\dat\1’
Nov 23 15:20:19 1 lcfd Cache Dir (cache_loc): ’C:\Tivoli\lcf\dat\1\cache’
Nov 23 15:20:19 1 lcfd Logging to (logfile): ’C:\Tivoli\lcf\dat\1\lcfd.log’ at level 2
Nov 23 15:20:19 1 lcfd Cache limit: ’20480000’
Nov 23 15:20:19 1 lcfd Cache size at initialization: ’0’
Nov 23 15:20:19 Q lcfd Entering lcf_run
Nov 23 15:20:19 1 lcfd ^Hmm... looks like you’re running NT 4.0  (build 1381).  Don’t
create a console.
Nov 23 15:20:20 2 lcfd Successful bind to lcfd preferred port 9494
Nov 23 15:20:20 2 lcfd Writing GCS file: C:\Tivoli\lcf\dat\1\last.cfg
Nov 23 15:20:20 1 lcfd node_login: listener addr ’0.0.0.0+9494’
Nov 23 15:20:20 2 lcfd No known gateways.
Nov 23 15:20:20 2 lcfd Trying other login listeners...
Nov 23 15:20:20 Q lcfd send_login_dgram: interval=30 attempts=2
Nov 23 15:20:20 Q lcfd net_usend of 402 bytes to 9.3.1.133+9494.  Bcast=0
Nov 23 15:20:20 Q lcfd send_login_dgram: waiting for reply. attempt 1 of 2
Nov 23 15:20:20 Q lcfd net_accept, handle=0x305f48
Nov 23 15:20:31 Q lcfd New connection from 9.3.1.133+2470
Nov 23 15:20:31 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:31 Q lcfd Leaving net_recv: bytes=714, (type=14 session=0)
Nov 23 15:20:31 Q lcfd recv: len=’714’ (code=’14’, session=’0’)
Nov 23 15:20:31 2 lcfd Writing GCS file: C:\Tivoli\lcf\dat\1\last.cfg
Nov 23 15:20:31 1 lcfd salmon is dispatcher 126 in region 1189622596
Nov 23 15:20:31 1 lcfd write login file ’lcf.dat’ complete
Nov 23 15:20:31 1 lcfd Logging into new gateway...
Nov 23 15:20:31 Q lcfd login_to_gw
Nov 23 15:20:31 Q lcfd login_gw -> 9.3.1.133+9494
Nov 23 15:20:31 2 lcfd Connecting to '9.3.1.133+9494'
Nov 23 15:20:31 Q lcfd net_send of 722 bytes, session 126
Nov 23 15:20:31 Q lcfd net_accept, handle=0x305f48
Nov 23 15:20:31 Q lcfd New connection from 9.3.1.133+2471
Nov 23 15:20:31 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:31 Q lcfd Leaving net_recv: bytes=730, (type=14 session=126)
Nov 23 15:20:31 Q lcfd recv: len='730' (code='14', session='126')
Nov 23 15:20:31 2 lcfd Writing GCS file: C:\Tivoli\lcf\dat\1\last.cfg
Nov 23 15:20:31 1 lcfd salmon is dispatcher 126 in region 1189622596
Nov 23 15:20:31 1 lcfd write login file 'lcf.dat' complete
Nov 23 15:20:31 1 lcfd final pid: 126
Nov 23 15:20:31 1 lcfd Login to gateway 9.3.1.133+9494 complete.
Nov 23 15:20:31 1 lcfd Ready.  Waiting for requests (0.0.0.0+9494). Timeout 120.
Nov 23 15:20:31 Q lcfd Entering Listener (running).
Nov 23 15:20:31 Q lcfd Entering net_wait_for_connection, timeout=-1 handle=0x305f48
Nov 23 15:20:31 Q lcfd New connection from 9.3.1.133+2472
Nov 23 15:20:31 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:31 Q lcfd Leaving net_recv: bytes=3592, (type=19 session=513819391)
Nov 23 15:20:31 2 lcfd CNTL_EP_CODESET
Nov 23 15:20:31 Q lcfd Entering Listener (running).
Nov 23 15:20:31 Q lcfd Entering net_wait_for_connection, timeout=-1 handle=0x305f48
Nov 23 15:20:32 Q lcfd New connection from 9.3.1.133+2473
Nov 23 15:20:32 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:32 Q lcfd Leaving net_recv: bytes=589, (type=0 session=513819392)
Nov 23 15:20:32 Q lcfd Entering send_methstat
Nov 23 15:20:32 Q lcfd Entering send_struct
Nov 23 15:20:32 Q lcfd net_send of 52 bytes, session 513819392
Nov 23 15:20:32 Q lcfd Leaving send_struct
Nov 23 15:20:32 Q lcfd Leaving send_methstat
Nov 23 15:20:32 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:32 Q lcfd Leaving net_recv: bytes=85, (type=7 session=513819392)
Nov 23 15:20:32 2 lcfd reading:
C:\Tivoli\lcf\dat\1\cache\bin\w32-ix86\endpoint\msg_bind.exe
Nov 23 15:20:32 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:32 Q lcfd Leaving net_recv: bytes=9226, (type=11 session=513819392)
Nov 23 15:20:32 Q lcfd Entering send_methstat
Nov 23 15:20:32 Q lcfd Entering send_struct
Nov 23 15:20:32 Q lcfd net_send of 52 bytes, session 513819392
Nov 23 15:20:32 Q lcfd Leaving send_struct
Nov 23 15:20:32 Q lcfd Leaving send_methstat
Nov 23 15:20:32 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:32 Q lcfd Leaving net_recv: bytes=95, (type=7 session=513819392)
Nov 23 15:20:32 2 lcfd reading: C:\Tivoli\lcf\generic\msg_cat\C\GatewayCatalog.cat
Nov 23 15:20:32 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:32 Q lcfd Leaving net_recv: bytes=1681, (type=11 session=513819392)
Nov 23 15:20:32 Q lcfd Entering send_methstat
Nov 23 15:20:32 Q lcfd Entering send_struct
Nov 23 15:20:32 Q lcfd net_send of 52 bytes, session 513819392
Nov 23 15:20:32 Q lcfd Leaving send_struct
Nov 23 15:20:32 Q lcfd Leaving send_methstat
Nov 23 15:20:32 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:32 Q lcfd Leaving net_recv: bytes=103, (type=7 session=513819392)
Nov 23 15:20:32 2 lcfd reading: C:\Tivoli\lcf\generic\msg_cat\fr_FR\GatewayCatalog.cat
Nov 23 15:20:32 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:32 Q lcfd Leaving net_recv: bytes=1910, (type=11 session=513819392)
Nov 23 15:20:32 Q lcfd Entering send_methstat
Nov 23 15:20:32 Q lcfd Entering send_struct
Nov 23 15:20:32 Q lcfd net_send of 52 bytes, session 513819392
Nov 23 15:20:32 Q lcfd Leaving send_struct
Nov 23 15:20:32 Q lcfd Leaving send_methstat
Nov 23 15:20:32 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:32 Q lcfd Leaving net_recv: bytes=103, (type=7 session=513819392)
Nov 23 15:20:32 2 lcfd reading: C:\Tivoli\lcf\generic\msg_cat\ja_JP\GatewayCatalog.cat
Nov 23 15:20:32 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:32 Q lcfd Leaving net_recv: bytes=2262, (type=11 session=513819392)
Nov 23 15:20:32 Q lcfd Entering send_methstat
Nov 23 15:20:32 Q lcfd Entering send_struct
Nov 23 15:20:32 Q lcfd net_send of 52 bytes, session 513819392
Nov 23 15:20:32 Q lcfd Leaving send_struct
Nov 23 15:20:32 Q lcfd Leaving send_methstat
Nov 23 15:20:32 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:32 Q lcfd Leaving net_recv: bytes=103, (type=7 session=513819392)
Nov 23 15:20:32 2 lcfd reading: C:\Tivoli\lcf\generic\msg_cat\pt_BR\GatewayCatalog.cat
Nov 23 15:20:32 Q lcfd Entering net_recv, receive a message
Nov 23 15:20:32 Q lcfd Leaving net_recv: bytes=1841, (type=11 session=513819392)
Nov 23 15:20:33 Q lcfd setting-up inherit fd. netfd=200
Nov 23 15:20:33 1 lcfd Spawning:
C:\Tivoli\lcf\dat\1\cache\bin\w32-ix86\endpoint\msg_bind.exe, ses: 1ea04300
Nov 23 15:20:33 Q lcfd Entering Listener (running).
Nov 23 15:20:33 Q lcfd Entering net_wait_for_connection, timeout=-1 handle=0x305f48
Nov 23 15:20:33 Q MethInit Entering mrt_run
Nov 23 15:20:33 Q MethInit argv: session_id=1ea04300
Nov 23 15:20:33 Q MethInit Communication timeout set: 120.
Nov 23 15:20:33 Q MethInit Entering comm_reconnect
Nov 23 15:20:33 Q MethInit inherited fd. return from net_associated_fd. ipc=3155416,
netfd=200
Nov 23 15:20:33 Q MethInit Entering run_impl
Nov 23 15:20:33 Q MethInit Entering send_methstat
Nov 23 15:20:33 Q MethInit Entering send_struct
Nov 23 15:20:33 Q MethInit net_send of 52 bytes, session 513819392
Nov 23 15:20:33 Q MethInit Leaving send_struct
Nov 23 15:20:33 Q MethInit Leaving send_methstat
Nov 23 15:20:33 Q MethInit waiting for input args
Nov 23 15:20:33 Q MethInit Entering net_recv, receive a message
Nov 23 15:20:33 Q MethInit Leaving net_recv: bytes=1577, (type=3 session=513819392)
Nov 23 15:20:33 2 MethInit Looking for method: write_html.
Nov 23 15:20:33 Q write_html calling method.
Nov 23 15:20:33 Q write_html method returned.
Nov 23 15:20:33 Q write_html send_results (max/len) 80/6
Nov 23 15:20:33 Q write_html Entering send_methstat
Nov 23 15:20:33 Q write_html Entering send_struct
Nov 23 15:20:33 Q write_html net_send of 60 bytes, session 513819392
Nov 23 15:20:33 Q write_html Leaving send_struct
Nov 23 15:20:33 Q write_html Leaving send_methstat
Nov 23 15:20:33 2 write_html Clean Shutdown write_html.
```

5.7.2.5 Other Log Files for Debugging TMA Problems

One of the most powerful debugging tools is the `wtrace` command, so we strongly recommend that you understand how to use the `wtrace` command. The following shows the usage of the `wtrace` command.

1. Set the trace option using the `odadmin trace` command. The following options can be used, but we strongly recommend that you use the `odadmin` trace `objcalls` option.

odadmin trace errors	Turns on error tracing.
odadmin trace services	Turns on service tracing.
odadmin trace objcalls	Turns on object call tracing.

2. Regenerate the problem.
3. Execute the `wtrace -jk $DBDIR > file_name`.
4. Turn off the trace with the `odadmin trace off`.

All methods issued by the management resources appear in the trace output, so it is very powerful to trace-detect the problem and understand the internal logic of the TMA environment as well.

5.7.2.6 How to Read wtrace

The following shows a sample output of the `wtrace` command when the Endpoint issues an upcall which requests a write message into the log file of the Sentry monitor.

```
loc-ic    81  M-hdq   Extern      806
      Time run:    [Wed 02-Dec 17:33:32]

      Object ID:   1189622596.2.605-
      Method:      LogToHostFile
      Principal:   nobody@lcf (0/-2)
      Path:        /aix4-r1/TME/SENTRY/sentry_gateway
      Input Data: (encoded):

             {
               "normal" "c:\" "166" "166" "166" "sp_Diskused_w" "salmon"
               {

                 {
                   1
                   [

                     {
                       "universal" "Disk space used" 10
                       {
                         "null" 0 false
                       }

                     }

                   ]

                 }

               }

               {

                 {
                   1
                   [

                     {
                       "SentEng" "(default result)" 126
                       {
                         "TMF_Types::_sequence_string_StringList" 19 false

                         {
                           0
                         }

                       }

                     }

                   ]

                 }

               }

               {

                 {
                   1
                   [

                     {
                       "SentEng" "" 134
                       {
                         "TMF_Types::_sequence_string_StringList" 19 false

                         {
                           0
```

```
                          }

                        }

                      }

                    ]

                  }

                }
                "" 912642004 ""
                {

                  {
                    1
                    [

                      {
                        "Sentry2_0" "(Mbytes)" 55
                        {
                          "null" 0 false
                        }

                      }

                    ]

                  }

                }
                "Universal" "diskused" 0 "salmon"
              }
               "/tmp/disk.log"  "kodiak"  "nobody"  "nobody"  0
loc-oc    81                        19
      Results: (encoded):
             "ok"
```

In this sample, you should focus on the following messages.

loc-ic	Refers to the sequential number of the method. In this case, the number is 81.
Time run	Refers to when the method was invoked. In this case, Dec 2nd 17:33:32.
Object ID	Refers to the object ID of the object in whose context the method was invoked.
Method	Refers to the name of the method. In this case, the LogToHostFile method is issued.
Principal	Refers to who issued the method. In this case, `nobody@lcf` issued the method.
Path	Refers to the location (path) of the method. In this case, the Endpoint issued the method which is located in /usr/local/Tivoli/bin/aix4-r1/TME/SENTRY/sentry_engine.

Input Data Refers to the contents of the data which is passed as the arguments to the method. In this case, this is the message which will be written into the log file.

loc-oc Refers to the result of the this method. In this case, the result of the LogToHostFile method is ok.

This downcall was logged into the gatelog file as well.

```
1998/12/02 17:33:32 +06: upcall start: from=9.3.1.193+1135, class=SentryGateway, method=LogToHostFil
```

The next figure shows a sample output of the `wtrace` command when the downcall was issued. This method is issued when the Software Distribution file package is distributed to the Endpoint.

```
loc-ic   206   M-ho    1-187     1.5K
      Time run:    [Sun 20-Dec 19:51:03]

      Object ID:   1588251808.1.348#TMF_ManagedNode::Managed_Node#
      Method:      fp_dist
      Principal:   root@ishii2.itsc.austin.ibm.com (0/0)
      Path:        /usr/local/Tivoli/bin/aix4-r1/TAS/MANAGED_NODE/fp_endpoint
      Trans Id:
             {
               1588251808:1,1588251808:1,18:88
             },

             {
               1588251808:1,1588251808:1,18:89
             },

             {
               1588251808:1,1588251808:1,18:103
             }
             #3
      Input Data: (binary) (dump suppressed)
```

As we mentioned, this sample shows that the `fp_dist` method is invoked. In this case, the principal field shows root user on the TMR Server (`root@ishii2.itsc.austin.ibm.com`) because this is the downcall. After the file package distribution was completed, the result of the `fp_dist` method (loc-ic 206) was logged into the `wtrace` output as follows:

```
loc-oc   206                     24
      Results: (encoded):

             {
               0
             }
              0
```

This downcall was logged into the gatelog file as well.

```
1998/12/20 19:51:05 +06: mdist: distribution ID = 1, method = fps_install, size = 0
```

Chapter 6. TMA and Tivoli Management Applications

In this chapter, we describe considerations for using the Tivoli Management Applications with the TMA. Basically, the TMA can replace the full Managed Node, except for in a few cases. In this chapter we will describe Tivoli application considerations regarding the TMA.

6.1 Implementation Considerations

There are a few considerations for using the Tivoli Management Applications, such as Distributed Monitoring, Software Distribution and so on, with the TMA. This section provides information related to Tivoli Management Applications running on the TMA.

6.1.1 Available Applications on the TMA

First of all, all of the core applications now provide support for the Tivoli Management Agent. This is a basic matter, but is very important. Although we will discuss some considerations here, you should refer to the *Release Notes* of the Tivoli Management Applications (every Tivoli product should include the *Release Notes*) for specific details.

The following table shows the status of the Endpoint support regarding the principal Tivoli Management Applications.

Table 40. Available Tivoli Management Applications on the TMA

Tivoli Management Application	Support for EP	Release
Tivoli Distributed Monitoring	NetWare EP only	V3.5.1 or V3.5.2
	Available	V3.6
Tivoli Inventory	Available	V3.6
Tivoli Remote Control	Available	V3.6
Tivoli Security Management	Available	V3.6
Tivoli Software Distribution	Available	V3.6
Tivoli Enterprise Console	Available	V3.6
Tivoli User Administration	Available	V3.6
Tivoli Modules	Not Available Yet	-

Note

This information is based on the *Release Notes* as of December 1998. You should confirm the latest *Release Notes* when you install the Tivoli Management Applications on the TMA.

6.1.2 Planning to Use Tivoli Management Applications with the TMA

Before we use the Tivoli Management Applications, we have to understand the restrictions and prerequisites of TMA implementation for your management system; then we will make a plan to implement the TMA and the Tivoli Management Applications. In this section, we discuss how to make a plan, what to consider, and so on. This information will be required by system engineers who develop management systems for an enterprise.

6.1.2.1 General Considerations for Using the TMA

The following restrictions and prerequisites should be clear before implementing the Tivoli Management Applications with the TMA.

The Managed Resource Must Remain a Managed Node:
Some Tivoli management resources still require the services provided by the full Managed Node. Therefore, if you configure the following management resources in the TMR, the machines on which they reside must be a Managed Node.

- TMR Server
- Endpoint gateways
- RIM Hosts
- MDist Repeaters
- Tivoli Enterprise Console Server
- Software Distribution Filepack Source Systems
- Systems Hosting Tivoli Plus Modules
- Systems Using the CLI Interface

The Version of the EP Manager, EP Gateway and Endpoint:
The LCF architecture is supported by both Versions 3.2 and 3.6 of Tivoli Management Framework, however, Version 3.6 of the Endpoint engine (`lcfd`) is different from Version 3.2 of the `lcfd`. Therefore, we have to be aware of the version of the EP Manager, EP gateway and Endpoint. Basically, the version of the Endpoint Manager and Endpoint gateway must be the same.

Therefore, at this time, the following three configuration schemes (see Figure 78 on page 213) are allowed when using the TMA.

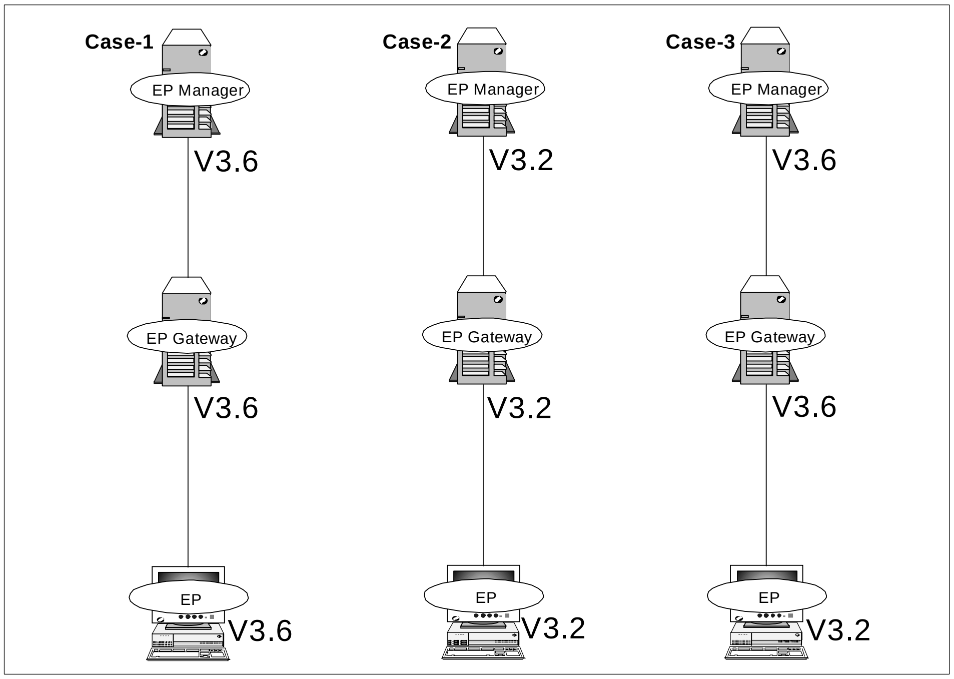


Figure 78. The Combination of the EP Manager, EP Gateway and EP

Case 1 This case shows Version 3.6 of the TMA configuration. We strongly recommend this configuration.

Case 2 This case shows Version 3.2 of the TMA configuration. As we mentioned before, the Endpoint Manager and Endpoint gateway must be the same version. Therefore, if you configure Version 3.2 of the Endpoint gateway, the Endpoint Manager and Endpoint should be the same version, V3.2, because Version 3.2 of the Endpoint gateway is not able to handle Version 3.6 of the Endpoint.

Case 3 This case is a little different from the other two cases. Version 3.6 of the Endpoint gateway is able to handle Version 3.2 of the Endpoint. In other words, Version 3.2 of the Endpoint can log in to Version 3.6 of the Endpoint gateway. However, Version 3.2 of the Endpoint will be updated automatically during the Endpoint login procedure if you have Auto Upgrade configured.

The Applications Should Be Installed on the Endpoint Gateway: The Tivoli Management Applications that will be used on Endpoints should be installed on the Endpoint gateway which the Endpoint is logged into. Therefore, we need to be aware of where the Endpoint is located when several Endpoint gateways are available in the TMR, when there is more than one TMR in the network, or when the available applications on each Endpoint gateway are different. In this situation, you should use the Endpoint policy (for instance, `select_gateway_policy`) and control the Endpoint login. We recommend that you install the same applications to all Endpoint gateways. This is the simplest solution, and prevents problems that could occur if an Endpoint should log into a gateway not supporting all of the applications.

6.1.3 Dataless Profile Manager and the TMA

The dataless profile manager is a new profile manager type associated with the dataless client (TMA). The dataless profile manager can have the following subscribers:

- Endpoints
- Managed Nodes
- PC Managed Nodes
- NIS Domains
- NetWare Managed Sites

As you can see in the list above, other profile managers cannot be subscribers to the dataless profile manager. Therefore, the dataless profile manager cannot be a branch node in a profile manager hierarchy. A dataless profile manager can only be a leaf node, that is, only have managed systems as subscribers.

6.1.3.1 What is the Difference between Dataless and Classic?

As you know, the TMA doesn't have an object database (.dbd file) locally; the information in the profile created in the dataless profile manager is not written to the database associated with the next level, even if it is a full Managed Node (refer to Figure 79 on page 215). Therefore, if you distribute the profile using the dataless profile manager to full Managed Nodes or Endpoints, the data is applied directly to the system.

The profile created in the classic profile manager can be locally modified because the information in the profile is written to the local database of the full Managed Node. However, the local modification is not a recommended customization because it makes profile management more complicated, and it also makes future migration from a full Managed Node to the TMA more difficult.

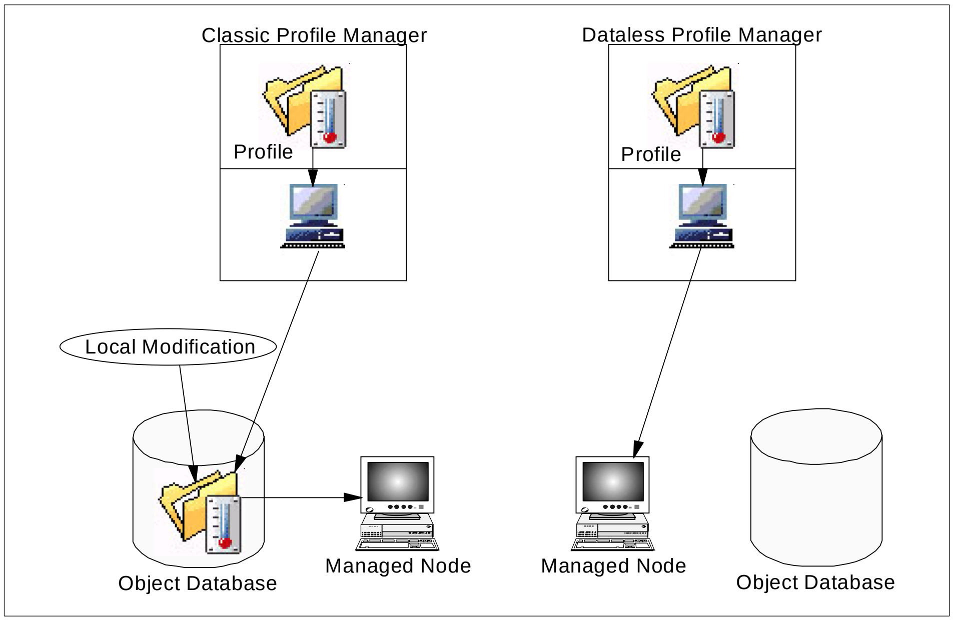


Figure 79. The Difference between Dataless and Classic Profile Managers

Creating a profile in a dataless profile manager and distributing it to a full Managed Node subscriber is the easiest way to observe the difference between the dataless and classic profile manager. The following test will show this:

1. Create the profile manager in dataless mode.
2. Create a Sentry profile in the dataless profile manager and define a full Managed Node (`ishii2`, in our example) as the subscriber, as shown in Figure 80 on page 216.

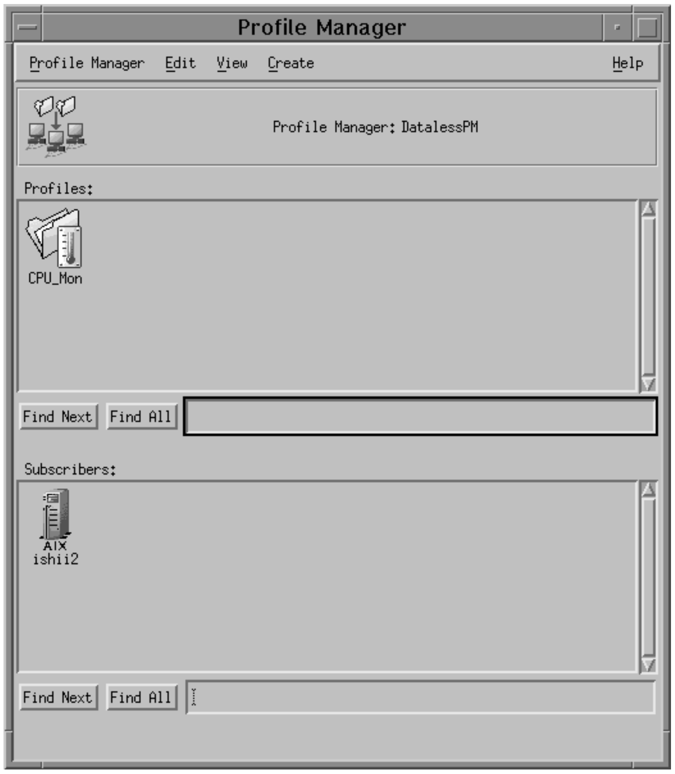


Figure 80. The Dataless Profile Manager and Full Managed Node Subscriber

3. Distribute the Sentry profile to the subscriber.
4. Look for the profile we distributed in the database of the subscriber as follows:

```
# wcd /Administrators/Root_ishii2-region/ishii2-region/ishii2
# wls -l
#
```

5. You will not see the profile object under the Managed Node, `ishii2`. This means the information in the dataless profile was not written to the database of the subscriber (`ishii2`). If you create the profile in a classic profile manager and distribute it to the full Managed Node, you can see the profile object, as in the following example.

D
R
A
F
T

```
# wls -l
1588251808.1.839#Sentry::All#   DM_UNIX@ishii2
```

As you can see from the above test, the dataless profile manager is developed for supporting Endpoint subscribers. However, almost all features except writing information to the database are the same as the classic profile manager, for example defining the profile, handling the profile, and so on.

Important

For this reason, we strongly recommend you use the dataless profile manager as follows:

- You should not create any profiles in the dataless profile manager.
- The dataless profile managers should be used as subscribers to classic profile managers for distributing profiles to the Endpoints.

This is the best way to avoid trouble regarding the dataless profile manager and the Endpoint.

6.1.4 Endpoint Method Cache Management

The Endpoint method cache management is very important to understanding the TMA implementation. The TMA gives you the framework functions that are necessary to perform management operations, such as those performed by Tivoli Management Applications, and these management operations are processed by calling a method on a managed resource. In the TMA environment, this design has not changed: the TMA invokes an Endpoint method for performing management operations on the TMA platform.

The Endpoint methods which will be used by the Endpoint are stored in the Endpoint gateway. When the TMA performs a management operation, the TMA automatically determines what is needed to perform the given management operation. If that Endpoint method already resides on the TMA, it immediately proceeds with the operation. If not, the TMA downloads the appropriate Endpoint method from the Endpoint gateway to the TMA with no operator intervention. In addition, the TMA downloads newer versions as updates are loaded on the Endpoint gateway. You can gain significant productivity advances with these management features because the Tivoli Management software is installed only once on the Endpoint Manager and Endpoint gateways, with updates performed automatically thereafter. The following figure shows how the TMA manages the Endpoint method cache.

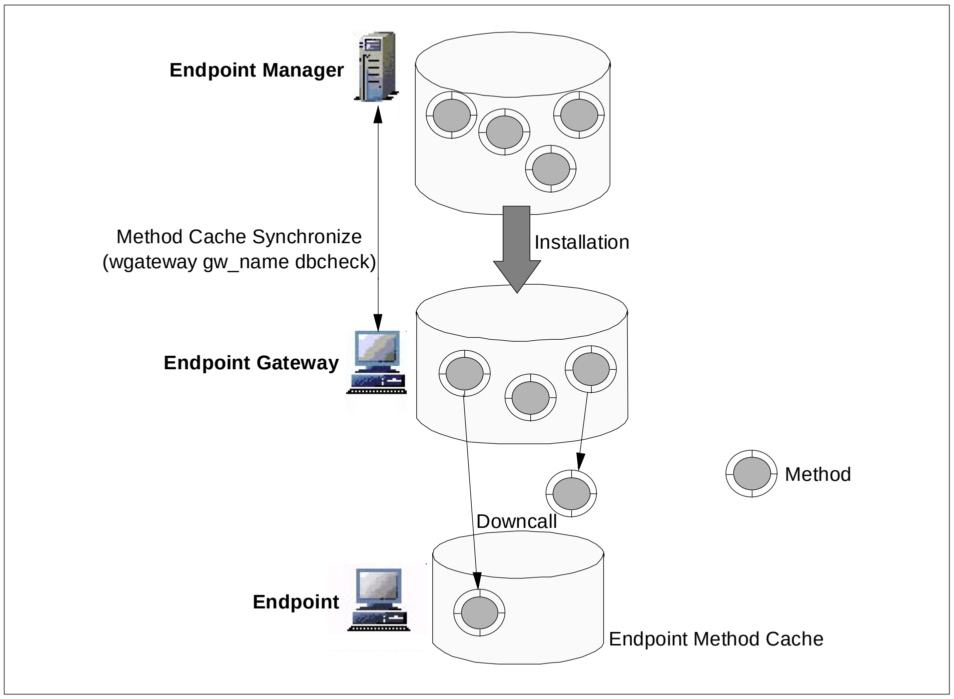


Figure 81. Endpoint Method Cache Management

By default, the Endpoint method cache exists under the C:\Tivoli\lcf\dat\1\cache directory, and each Endpoint method actually corresponds to the *.exe file in the PC environment. The Endpoint gateway stores the Endpoint method under the /usr/local/Tivoli/bin/lcf_bundle directory and also stores Endpoint method information in the Endpoint gateway database ($DBDIR/gwdb.bdb file). This information can be synchronized with the information on the Endpoint Manager ($DBDIR/imdb.bdb file) using the `wgateway gw_name dbcheck` command.

The following summary describes how the Endpoint manages the Endpoint method cache.

- Once the Endpoint method is stored in the Endpoint method cache, the Endpoint uses it even if the system is rebooted.
- Once the Endpoint method is downloaded to the Endpoint method cache, the Endpoint does not download the same version of the Endpoint method.

- If the Endpoint detects the available version is greater than the current version of the Endpoint method, the Endpoint downloads the newer method automatically.
- When the Endpoint downloads the method, the related methods defined in the dependency set are also downloaded at the same time. The dependency set is a list of other files, modules or commands that are required for the correct operation of the method.

6.1.5 Endpoint Methods and Tivoli Management Applications

The Tivoli Management Applications invoke methods using upcall and downcall to perform their functions. Which method will be used by a Tivoli Management Application? The dependency manager handles the methods for each Tivoli Management Application. The dependency manager exists as a class object in the Tivoli object database and plays a very important role in implementing the LCF architecture. We can understand the relationship between the methods and the Tivoli Management Applications by checking the dependency manager. We provide detailed information regarding the dependency manager in Chapter 8, “Overview of TMA Internals and Application Development” on page 281; so at this time we will only introduce the relationship between the methods and the Tivoli Management Applications.

Normally, the dependency set is added to the dependency manager when we install the Tivoli Management Application (but it is not mandatory, it depends on the application design). We installed some Tivoli Management Applications for checking the dependency during the project, and the following table (Table 41 on page 219) shows the relationship between the dependency set and the Tivoli Management Applications.

Table 41. The Dependency Set for Each Application

Dependency Set	Tivoli Management Application
LCFDepList	Distributed Monitoring
courier_lcf	Software Distribution
GroupManagement	User Administrator
UserManagement	

To make sure, you can use the `wlookup` and `wdepset` command as follows.

```
# wlookup -ar DependencyMgr
GroupManagement 1588251808.1.726#Depends::Mgr#
LCFDepList      1588251808.1.621#Depends::Mgr#
NtLcfInst_depset        1588251808.1.523#Depends::Mgr#
UserManagement  1588251808.1.721#Depends::Mgr#
courier_lcf     1588251808.1.810#Depends::Mgr#
msg-bind-catalogs       1588251808.1.516#Depends::Mgr#
task-lcf-base   1588251808.1.513#Depends::Mgr#
task-spawn16    1588251808.1.514#Depends::Mgr#
task-spawn32    1588251808.1.515#Depends::Mgr#
```

In this case, we invoked the `wlookup` command. In the TMR, we installed Distributed Monitoring, Software Distribution and User Administrator. If you would like to know which methods are included in the dependency set, just invoke the `wdepset` command as follows:

```
# wdepset -v 1588251808.1.621
```

The above example introduces how to display the methods set for Distributed Monitoring. The output of the `wdepset` command can be very large, so we won’t show it now. These methods should be located under the /usr/local/Tivoli/bin/lcf_bundle directory and loaded to the method cache in the Endpoint where the application is called for the first time (for instance, when the Sentry profile is distributed to the Endpoint subscriber for the first time). The following figure (Figure 82 on page 221) shows the relationship between the Endpoint methods and the dependency manager.

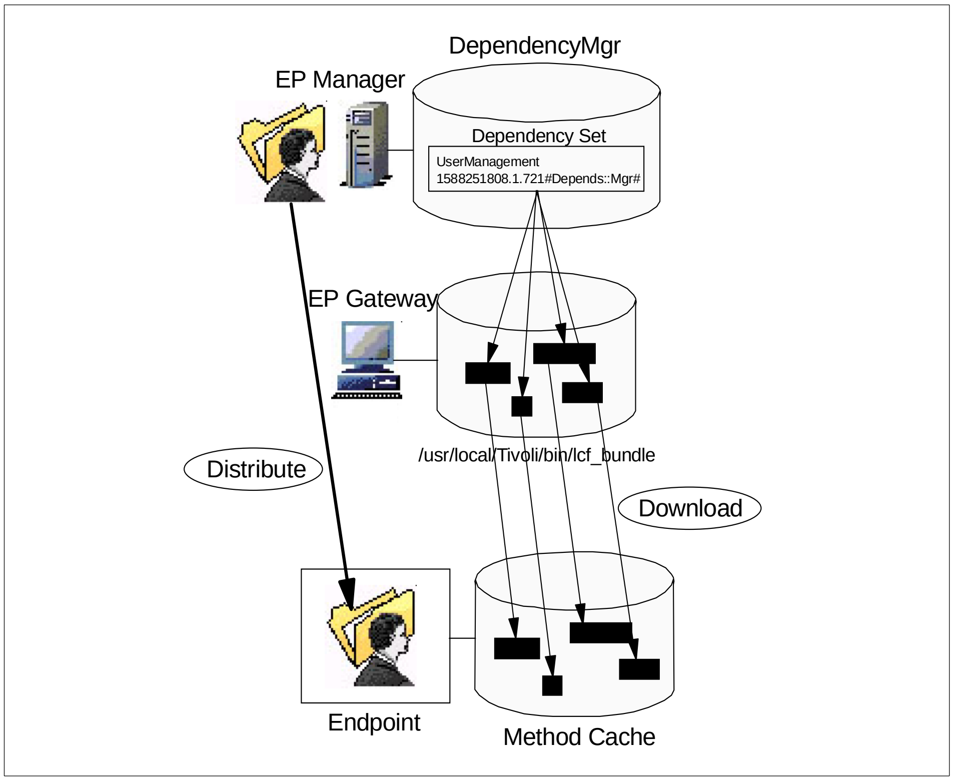


Figure 82. The Dependency Manager and Endpoint Methods

6.1.6 Boot_method and Tivoli Management Applications

Have you ever used the Sentry monitor on the Endpoint? If so, have you thought about why the Sentry engine boots automatically every time when the `lcfd` daemon starts? You might look at the Startup menu or the Services menu of the Windows NT machine (for instance, the Windows NT Endpoint). However, you cannot find any definition regarding the Sentry engine in the Startup and Services menus. How does the Sentry engine start automatically every time the `lcfd` starts?

The `lcfd` daemon has a boot_method list for starting the methods automatically when the `lcfd` boots. The Sentry engine of Distributed Monitoring is a typical application for using the boot_method list. Therefore, we introduce an example of how the Sentry engine starts automatically using the boot_method.

When you distribute the Sentry profile, including a monitor to an Endpoint for the first time, the boot_method list is created for booting the Sentry engine automatically every time the lcfd boots. Each boot_method list has a tag for distinguishing each list from others, so the tag is a kind of an ID. By default, distributed monitoring uses a tag named Sentry_Boot_1. You can see this by using the wtrace command during the lcfd booting. The following output of the wtrace shows that the Endpoint Manager added the new boot_method named Sentry_Boot_1 for the Sentry engine using the add_boot_method method (We will talk about the methods under the Endpoint Manager in Chapter 7, “Advanced Knowledge of the TMA” on page 251). This process should be performed when you distribute the Sentry profile to the Endpoint for the first time.

```
loc-ic   299 M-hdoq     2-61      145
      Time run:    [Wed 02-Dec 17:26:06]

      Object ID:   1189622596.1.517#TMF_LCF::EpMgr#
      Method:      add_boot_method
      Principal:   nobody@lcf (-2/-2)
      Path:        __epmgr_implid
      Input Data: (encoded):

             {
               1
               [
                 "1189622596.239.508+#TMF_Endpoint::Endpoint#"
               ]

             }
              "Sentry_Boot_1"  "1189622596.1.565"  "boot_engine"
```

You are also able to confirm the contents of the boot_method list using the wep command as follows:

```
# wep boot_method list Sentry_Boot_1 1189622596.239.508+
Boot Method(s) for Endpoint 1189622596.239.508+
Tag                      Prototype Object     Method Name
Sentry_Boot_1            1189692596.1.565     boot_engine
```

The boot_method list can be used for any user applications that run on the Endpoint as a daemon process, such as the Sentry engine. To define a new boot_method, you can use the wep boot_method add command. The boot_method list would be used for a daemon-like process running on the Endpoint and needs to be executed automatically every time the lcfd boots. The boot_method information of the Endpoint is managed by the Endpoint Manager.

6.1.7 Database Backup and Tivoli Management Applications

The backup process is very important in the Tivoli Management environment in case the data of the Tivoli object database is corrupted. The most reliable way to recover it is to restore the data from a backup. As you know, the TMA doesn't have an object database, so the TMA doesn't have any data related to the applications running on the TMA. This is the reason the TMA needs the dataless profile manager. So where is the data related to the Tivoli Management Applications and its configurations? The data still exists in the Tivoli object database (odb.bdb file) even if the applications run on the TMA. Therefore, normally, information defined in the profiles is backed up using the `wbkupdb` command.

Note

Most of the Tivoli Management Application's configuration can be backed up by the `wbkupdb` command; however, the Distributed Monitoring application is a little different from others because of the daemon-like process, the Sentry engine, on each TMA. We will talk about database backup and Sentry monitors in Section 6.2.8, "Database Backup and Sentry Monitors on the TMA" on page 242.

6.2 Upcall Applications and the TMA

The Tivoli Management Applications that run on an Endpoint can be classified into two categories: upcall-oriented applications and downcall-oriented applications. Amazingly enough, upcall oriented applications mainly use upcalls, and downcall oriented applications mainly use downcalls.

Normally, upcall applications are more complicated than downcall applications in their implementation. Distributed Monitoring is a typical upcall-oriented application and provides a good example. In this section, we focus on the Distributed Monitoring and introduce considerations and advantages of how Distributed Monitoring is implemented on the TMA, how Distributed Monitoring issues upcalls, as well as the interaction between Distributed Monitoring and the TMA.

6.2.1 Distributed Monitoring and the TMA

Availability applications, for example Distributed Monitoring, Tivoli Enterprise Console and so on, are the most typical applications for Tivoli customers because every system administrator would like to monitor the status and condition of managed systems. Therefore, many Tivoli customers will use Distributed Monitoring or Tivoli Enterprise Console with TMA. The TMA can replace a full Managed Node; however, there are some differences between the applications on the TMA and on the full Managed Node. It is very important to understand the differences, the advantages, and the considerations for implementing upcall-oriented applications, such as Distributed Monitoring, on the TMA.

6.2.2 Sentry Engine on TMA

As you know, Distributed Monitoring contains a daemon-like process called the Sentry engine. It is very unique. The Sentry engine runs on the node that we are monitoring using a Sentry monitor. Version 3.6 of Tivoli implements the LCF architecture, and the Sentry engine still exists even on the Endpoints. To see this, use the Task Manager of Windows NT, and you can find the Sentry engine as one process, `dm_ep_engine.exe`. This is shown in Figure 83 on page 225. The Sentry engine running on the Endpoint (`dm_ep_engine.exe`) provides almost the same function as the `sentry_engine` process running on the full Managed Node. Basically, the actions that the Sentry engine performs when the Sentry engine detects specified conditions are the following, and these can be defined in the Sentry profile.

- Sending Tivoli Notices
- Displaying a Pop-up Message
- Changing Indicator Icon
- Sending e-mail
- Logging to File
- Running Program
- Sending Tivoli Enterprise Console Event

The interesting thing here is that these actions are performed by the upcall from the Sentry engine running on the Endpoint. This is very different from other Tivoli Management Applications. We introduce the information regarding how the Sentry engine performs the above actions using the upcall and so on in Section 6.2.7, “Understanding Distributed Monitoring Behavior with TMA” on page 238.

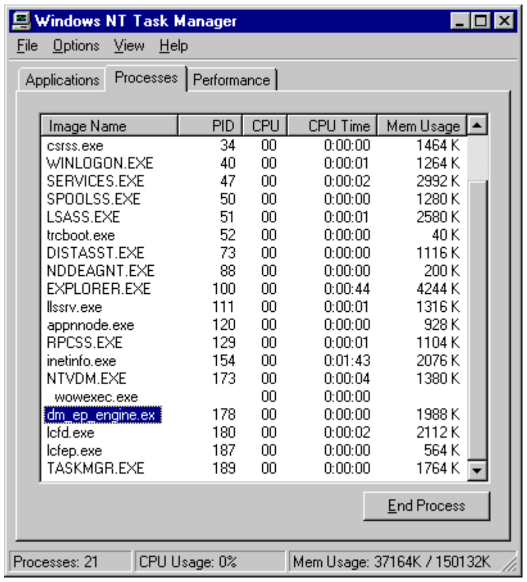


Image Name	PID	CPU	CPU Time	Mem Usage
csrss.exe	34	00	0:00:00	1464 K
WINLOGON.EXE	40	00	0:00:01	1264 K
SERVICES.EXE	47	00	0:00:02	2992 K
SPOOLSS.EXE	50	00	0:00:00	1280 K
LSASS.EXE	51	00	0:00:01	2580 K
trcboot.exe	52	00	0:00:00	40 K
DISTASST.EXE	73	00	0:00:00	1116 K
NDDEAGNT.EXE	88	00	0:00:00	200 K
EXPLORER.EXE	100	00	0:00:44	4244 K
llssrv.exe	111	00	0:00:01	1316 K
appnnode.exe	120	00	0:00:00	928 K
RPCSS.EXE	129	00	0:00:01	1104 K
inetinfo.exe	154	00	0:01:43	2076 K
NTVDM.EXE	173	00	0:00:04	1380 K
wowexec.exe		00	0:00:00	
dm_ep_engine.ex	178	00	0:00:00	1988 K
lcfd.exe	180	00	0:00:02	2112 K
lcfep.exe	187	00	0:00:00	564 K
TASKMGR.EXE	189	00	0:00:00	1764 K

Processes: 21 CPU Usage: 0% Mem Usage: 37164K / 150132K

Figure 83. The Sentry Engine Process in the NT Task Manager

6.2.3 Sentry Gateway Process

To support the Endpoint, the distributed monitoring provides a new process, `sentry_gateway`. The `sentry_gateway` process runs on the Endpoint gateway machine and plays the role of the translator between the Endpoint and Endpoint gateway, so that the `sentry_gateway` process handles the requests (upcalls) sent from the Sentry engine (`dm_ep_engine.exe`) on the Endpoint. The next figure (Figure 84 on page 226) shows all the processes related to Distributed Monitoring in the TMA environment.

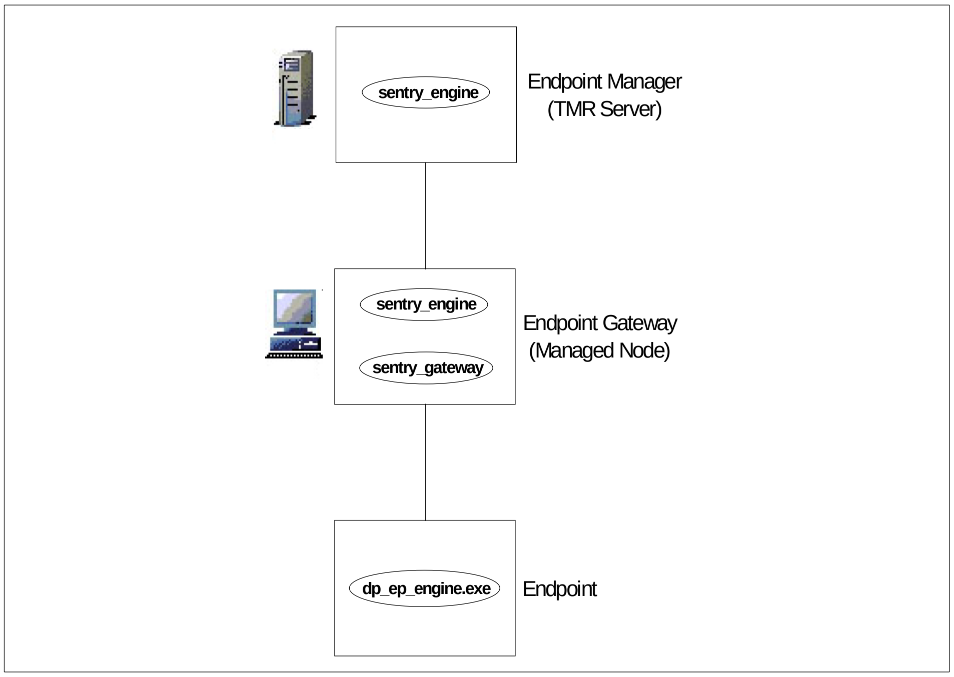


Figure 84. The Sentry Engine Processes

Later on, we will explain the role of each process related to distributed monitoring.

6.2.4 Booting Sentry Engine

When the Sentry engine boots, how does the Sentry engine recognize the monitor already defined and start it? It is a little complicated, but it provides a good example for understanding the process provided by Distributed Monitoring. We describe the booting procedure in Figure 85 on page 227.

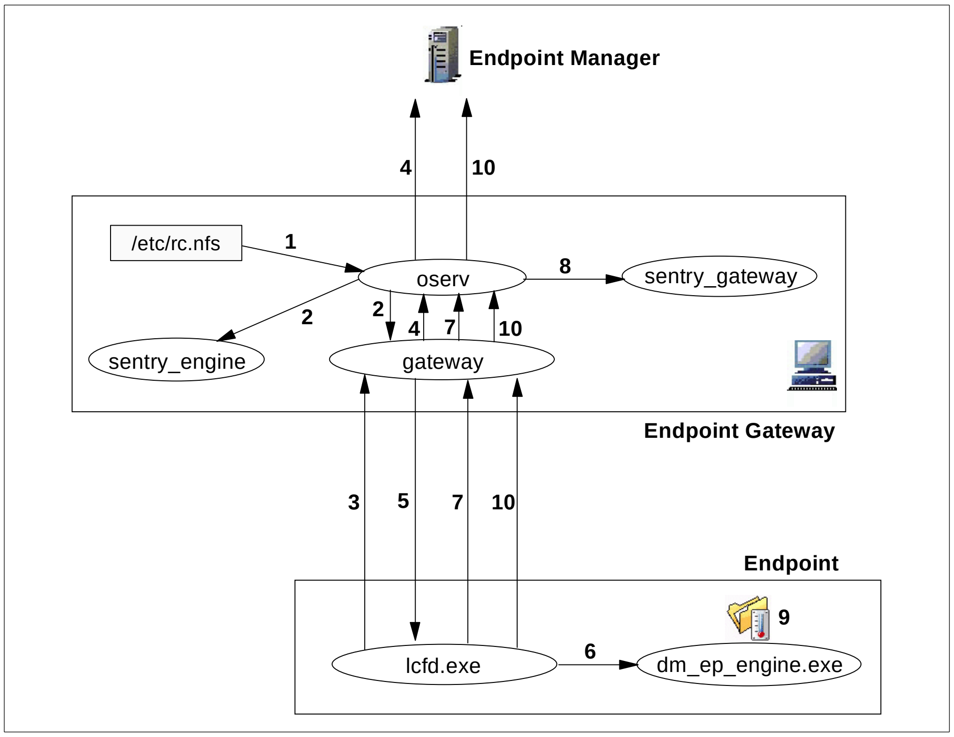


Figure 85. The Booting Procedure of the Sentry Engines

1. When the Endpoint gateway machine boots, the `oserv` daemon should be booted automatically from the definition in the /etc/rc.nfs file (for UNIX machine) or services (for NT machine).
2. The oserv daemon also starts the process defined in the `boot_method` list. Normally, if you have installed Distributed Monitoring to the Endpoint gateway machine, the Endpoint gateway process (`gateway`) and the Sentry engine (`sentry_engine`) should be defined in the boot_method by default, so that both processes are booted by the `oserv` daemon at this time.
3. After the Endpoint gateway becomes available, the Endpoint attempts to perform the login to the appropriate Endpoint gateway (the detailed login procedure is omitted here).
4. After the Endpoint login is completed, the Endpoint gateway attempts to invoke a method on the Endpoint Manager remotely for checking the boot_method list of the Endpoint. If you have already installed any Sentry monitors onto the Endpoint, the Sentry engine should be defined in the boot_method list of the Endpoint. Then the Endpoint gateway gets the

information regarding the `boot_method` list of the Endpoint and recognizes that the Sentry engine already exists on the Endpoint.

5. The Endpoint gateway invokes the `boot_engine` method using the downcall to start the Sentry engine on the Endpoint.
6. The Sentry engine (`dm_ep_engine.exe`) on the Endpoint is started by the downcall from the Endpoint gateway at this time.
7. When the Sentry engine boots on the Endpoint, the Endpoint invokes the `GetCollList` on the Endpoint gateway using an upcall remotely for collecting the Sentry monitor information.
8. Then, if the `sentry_gateway` process is not running, the `sentry_gateway` process is booted automatically by the `oserv` daemon using the Basic Object Adapter function and provides the information regarding the enabled Sentry monitors.
9. The Endpoint gets the information regarding the enabled Sentry monitors. Then the enabled Sentry monitors begin to monitor with the defined configurations.
10. Finally, the Endpoint invokes the `add_boot_method` method on the Endpoint Manager using an upcall and defines the Sentry engine in the `boot_method` list of the Endpoint again for the next booting.

To confirm this procedure for yourself, you can use the `wtrace`, `gatelog` or `lcfd.log` files. In other words, most of the TMA behaviors can be investigated by examining this information. For more information regarding the `wtrace` and log files, refer to Chapter 5, “Anatomy of TMA Behavior” on page 135.

6.2.5 Distributed Monitoring Method Cache

By default, Distributed Monitoring has a cache exclusively for storing methods under the following directories:

NT	C:\Tivoli\lcf\dat\1\lcf\sentry
UNIX	/opt/Tivoli/lcf/dat/1/LCF/sentry

One of the reasons why Distributed Monitoring has its own exclusive cache is that the Sentry engine could use a lot of methods, and the total size of these methods could be large (approximately 3 MB for a Windows NT Endpoint).

After distributing the Sentry profile to the Endpoint, you can use the Windows NT Explorer to see the methods loaded into the method cache of Distributed Monitoring. These methods should be the same as the contents of the dependency set we explained in Section 6.1.5, “Endpoint Methods and Tivoli Management Applications” on page 219.

6.2.6 Distributed Monitoring and Endpoint Methods

As we mentioned, the Distributed Monitoring actions we configure in the Sentry profile would normally use the upcall. This means the Tivoli Enterprise Console event or Tivoli notices would be sent from the Endpoint gateway, not from the Endpoint. This is a very important point for understanding upcall applications, such as Distributed Monitoring. We introduce some examples of how Distributed Monitoring invokes a method on the Endpoint gateway.

6.2.6.1 Sending Tivoli Notices

When the Endpoint sends a Tivoli notice to the TMR server, the Endpoint issues an upcall by invoking the `SendNotice` method stored in the Endpoint gateway. Then the Tivoli notice is sent from the Endpoint gateway to the TMR server. However, the sending procedure is a little different between the first Tivoli notice and others. We introduce both cases as follows:

Sending the Tivoli Notice for the First Time:

In this case, we assume the Sentry profile that includes the definition of sending Tivoli notices is distributed to the Endpoint for the first time. To send the Tivoli notice to the TMR server, the following methods would be invoked.

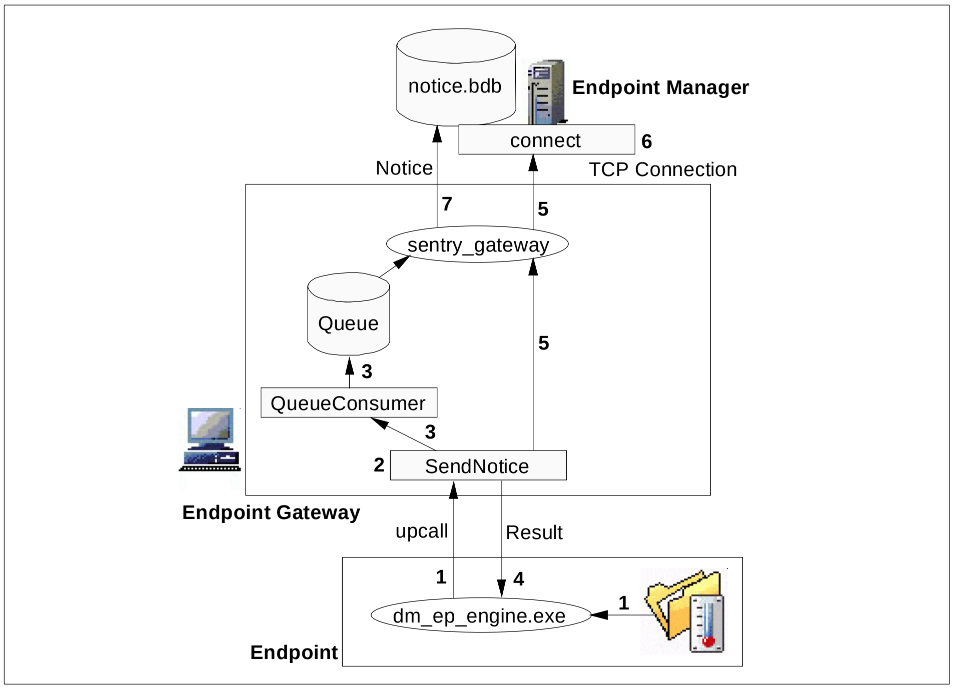


Figure 86. Sending the Tivoli Notice for the First Time

1. When the Sentry monitor detects the condition that should send the Tivoli notice, the Endpoint issues the upcall for invoking the `SendNotice` method on the Endpoint gateway.
2. The `SendNotice` method is invoked by the upcall from the Endpoint. Then the other related methods are invoked. To get the monitor collection information from the Endpoint Manager, the `_get_collection` method is invoked by the Endpoint many times, repeatedly.
3. The `QueueConsumer` method is invoked at that time, and it plays the role of the queue for the Tivoli notices.
4. After the `QueueConsumer` method is invoked without error, the Endpoint gateway returns the result of the upcall issued by the Endpoint.
5. After returning the result of the upcall, the `connect` method on the TMR server is invoked remotely from the Endpoint gateway for establishing the TCP session.
6. The `connect` method is invoked by the Endpoint gateway, and then the TCP session is established. This TCP session would be used for sending the Tivoli notices from the Endpoint to the TMR server.
7. The Endpoint gateway sends the Tivoli notice to the TMR server using the TCP session.

Sending Tivoli Notices in Normal Cases:
If more than one Tivoli notice has been sent, the sending procedure would be different after the first Tivoli notice. The following figure shows the sending procedure in this case.

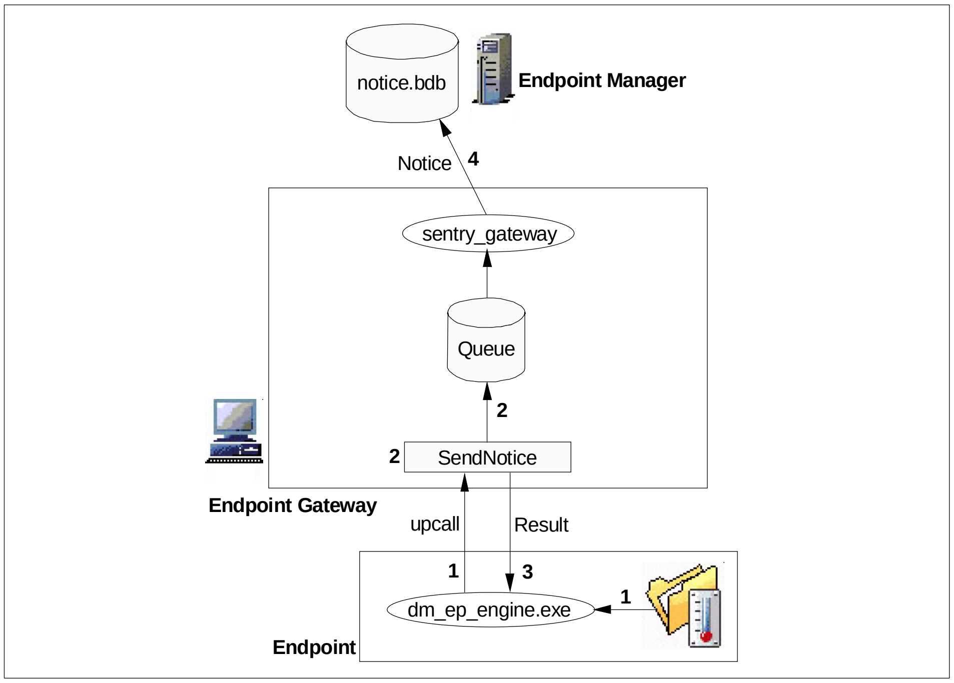


Figure 87. Sending Tivoli Notices Routinely

1. When the Sentry monitor detects the condition that should send the Tivoli notice, the Endpoint issues the upcall for invoking the `SendNotice` method on the Endpoint gateway.
2. The `SendNotice` method is invoked by the upcall from the Endpoint. Then the information of the Tivoli notice would be transferred to the queue.
3. After the `SendNotice` method is invoked without error, the Endpoint gateway returns the result of the upcall to the Endpoint.
4. The Tivoli notice transferred to the queue would be sent by the `sentry_gateway` process to the TMR server.

6.2.6.2 Sending Tivoli Enterprise Console Events

When the Endpoint sends a T/EC event to the T/EC server, the Endpoint issues an upcall for invoking the `SendBetterTECevent` method stored in the Endpoint gateway. Then the T/EC event is sent from the Endpoint gateway to the T/EC server. However, the sending procedure is a little different between the first T/EC event and other events, which are also sending Tivoli notices. We introduce both cases as follows:

Sending the T/EC Event for the First Time:

In this case, we assume the Sentry profile that includes the definition of the sending TEC events is distributed to the Endpoint for the first time. To send the TEC event to the TEC server, the following methods would be invoked.

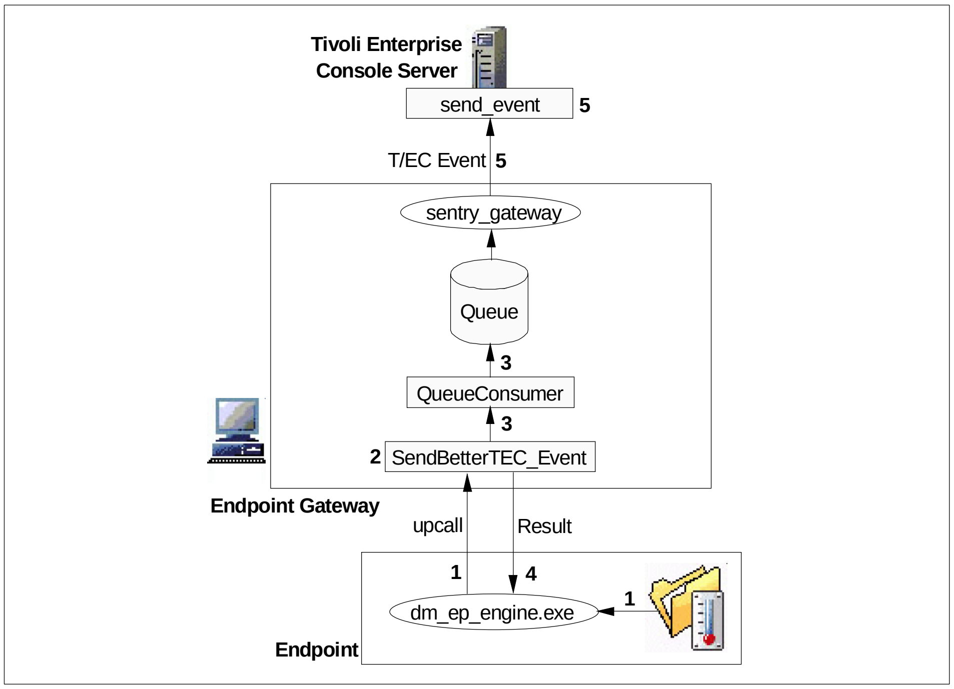


Figure 88. Sending the T/EC Event for the First Time

1. When the Sentry monitor detects a condition that should generate a TEC event, the Endpoint issues the upcall for invoking the `SendBetterTEC_Event` method on the Endpoint gateway.
2. The `SendBetterTEC_Event` method is invoked by the upcall from the Endpoint. Then the other related methods are invoked. To get the monitor collection information from the Endpoint Manager, the `_get_collection` method is invoked by the Endpoint many times in this sequence.
3. The `QueueConsumer` method is invoked, and it plays the role of the queue for the TEC events as well as for Tivoli notices.
4. After the `QueueConsumer` method is invoked without error, the Endpoint gateway returns the result of the upcall to the Endpoint.
5. After returning the result of the upcall, the `send_event` method on the TEC server is invoked remotely from the Endpoint gateway for sending the

TEC event. As a result, the TEC event is sent from the Endpoint gateway to the TEC server.

Sending TEC events routinely:

If more than one TEC event has been sent, the sending procedure would be different from the first TEC event. The following figure shows the sending procedure in this case.

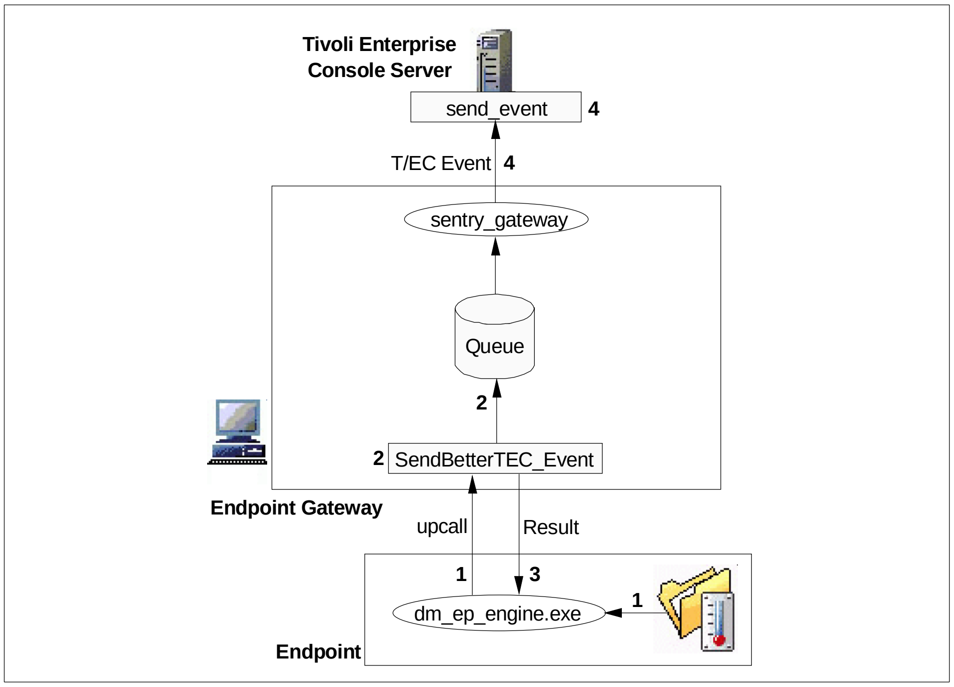


Figure 89. Sending TEC Event Routinely

1. When the Sentry monitor detects the condition that should send the TEC event, the Endpoint issues the upcall for invoking the `SendBetterTEC_Event` method on the Endpoint gateway.
2. The `SendBetterTEC_Event` method is invoked by the upcall from the Endpoint. Then the information of the TEC event is transferred to the queue.
3. After the `SendBetterTEC_Event` method is invoked without error, the Endpoint gateway returns the result of the upcall to the Endpoint.
4. After returning the result of the upcall, the `send_event` method on the TEC server is invoked remotely from the Endpoint gateway for sending the

TEC event. As a result, the TEC event is sent from the Endpoint gateway to the TEC server.

Note

As you can see, all Tivoli notices and TEC events are sent from the Endpoint gateway, not from the Endpoint. It is the same for the Tivoli Event Adapter (NT eventlog file adapter). The Endpoint just issues an upcall for sending a Tivoli notice or TEC event. It is very important to understand the relationship between the Endpoint and Endpoint gateway.

6.2.6.3 Logging to File

This action is a little different from the other actions because the Sentry monitor can write logs to each managed resource, Endpoint Manager (TMR server), Endpoint gateway (Managed Node), and Endpoint (local disk). In this section, we introduce each case of logging to files as follows.

Logging to Endpoint Manager:

In this case, we assume the Sentry monitor writes the log to the log file located on the Endpoint Manager. The following figure shows the logging procedure.

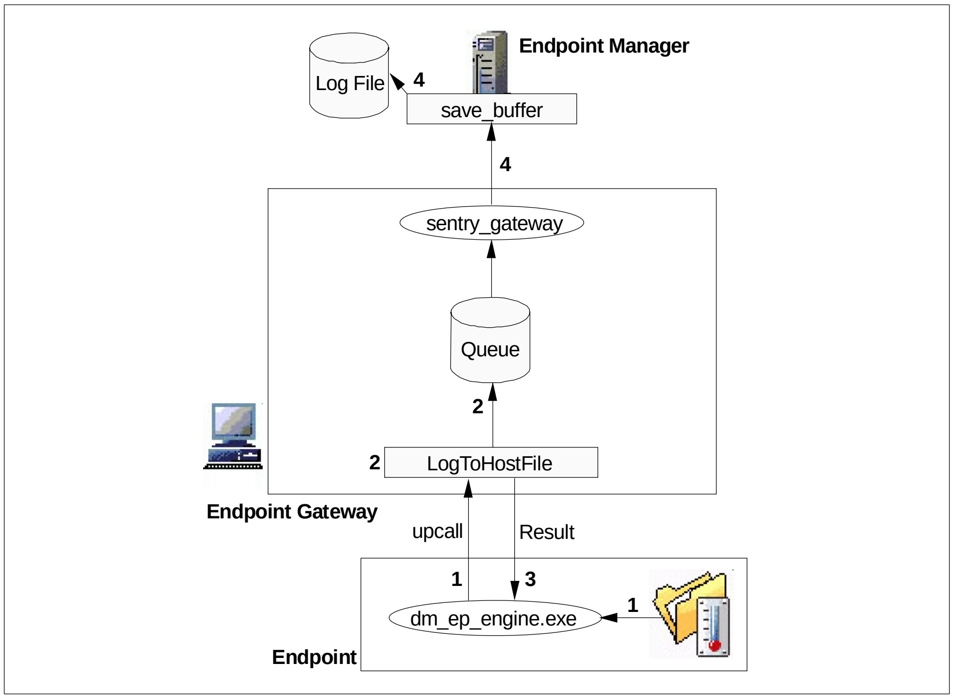


Figure 90. Logging to Endpoint Manager

1. When the Sentry monitor detects the condition that requires it to write log into the log file located in the Endpoint Manager, the Endpoint issues the upcall for invoking the `LogToHostFile` method on the Endpoint gateway.
2. The `LogToHostFile` method is invoked by the upcall from the Endpoint. Then the information for logging is transferred to the queue. (If the `QueueConsumer` method has not been invoked to get the monitor collection information from the Endpoint Manager, the `_get_collection` method is invoked by the Endpoint many times, and then the `QueueConsumer` method is invoked, and then the Tivoli notice is sent to the server.)
3. After the `LogToHostFile` method is invoked without error, the Endpoint gateway returns the result of the upcall to the Endpoint.
4. After returning the result of the upcall, the `save_buffer` method on the TMR server is invoked remotely from the Endpoint gateway for writing the log to the TMR server. As a result, the log is written to the log file on the Endpoint Manager.

Logging to the Endpoint Gateway:
In this case, we assume the Sentry monitor writes logs to the log file located on the Endpoint gateway. The following figure shows the logging procedure.

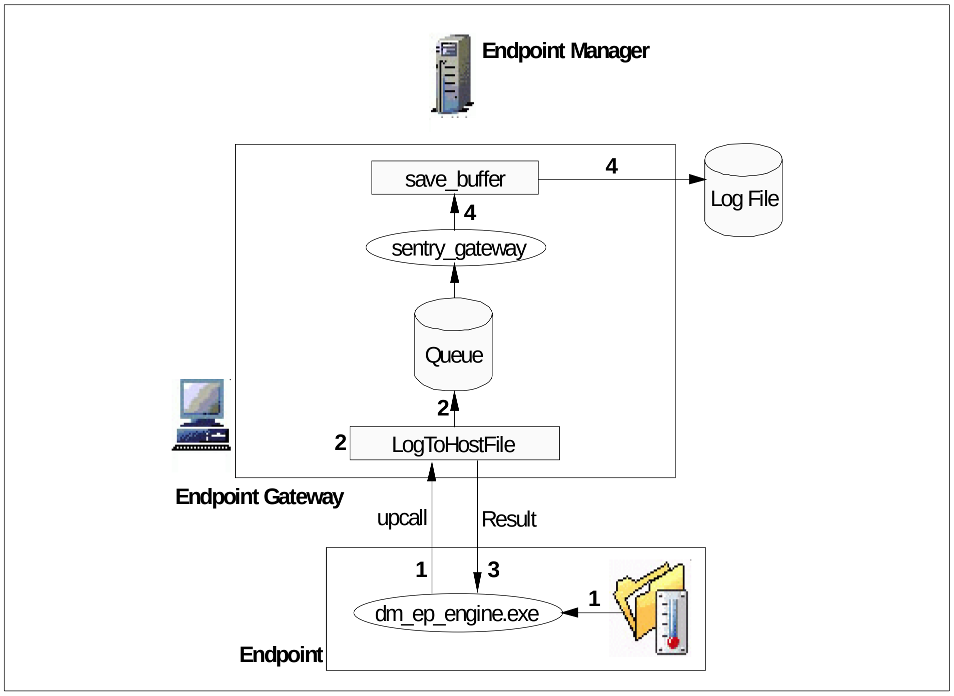


Figure 91. Logging to the Endpoint Gateway

1. When the Sentry monitor detects the condition that requires it to write log information into the log file located in the Endpoint gateway, the Endpoint issues the upcall for invoking the `LogToHostFile` method on the Endpoint gateway.
2. The `LogToHostFile` method is invoked by the upcall from the Endpoint. Then the information to be logged is transferred to the queue. (If the `QueueConsumer` method has not been invoked to get the monitor collection information from the Endpoint Manager, the `_get_collection` method is invoked by the Endpoint many times, and then the `QueueConsumer` method is invoked at that timing as well as sending Tivoli notices.)
3. After the `LogToHostFile` method is invoked without error, the Endpoint gateway returns the result of the upcall to the Endpoint.
4. After returning the result of the upcall, the `save_buffer` method on the Endpoint gateway is invoked locally for writing the log to the Endpoint

gateway. As a result, the log is written to the log file on the Endpoint gateway.

Logging to Endpoint:
In this case, we assume the Sentry monitor writes the log to the log file located on the Endpoint. The following figure shows the logging procedure.

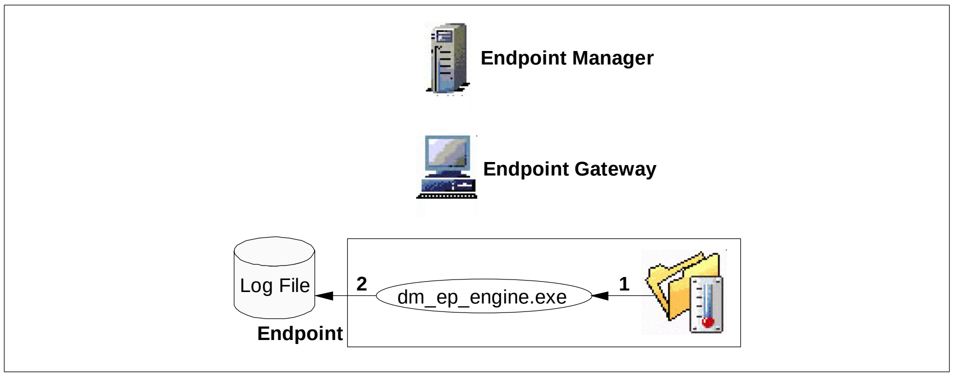


Figure 92. Logging to Endpoint

1. When the Sentry monitor detects the condition that requires it to write log into the log file located in the Endpoint, the Sentry engine process (dm_ep_engine.exe) on the Endpoint attempts to write the log to the log file.
2. The log is written to the local log file by the Sentry engine. No upcall is issued in this case.

6.2.6.4 Role of the Sentry Gateway

As you can see, the sentry_gateway process plays the role of the proxy in the TMA environment. When the sentry_gateway receives the request from the Endpoint by an upcall, the sentry_gateway invokes the appropriate method on the full Managed Node, for example the TEC server or TMR Server, instead of the Endpoint. After the method invocation, the sentry_gateway returns the result to the Endpoint. It acts as the proxy of the Endpoint Sentry engine. This implementation is very important for understanding how Distributed Monitoring supports the TMA.

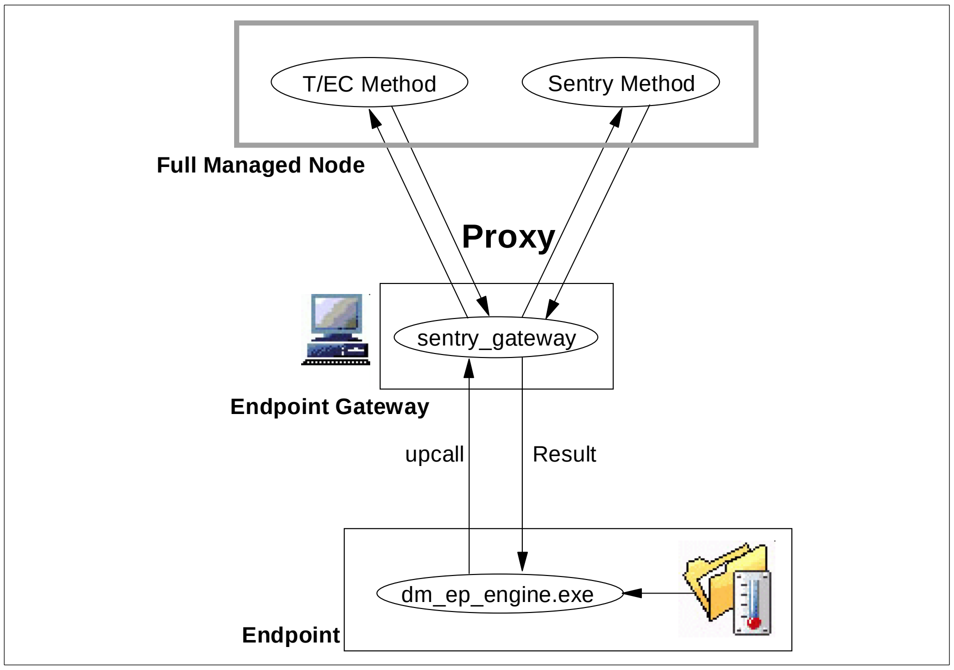


Figure 93. The Role of the sentry_gateway Process

6.2.7 Understanding Distributed Monitoring Behavior with TMA

In this section, we assume some problem situations, for example the network goes down or the Endpoint gateway becomes unavailable. In this situation, we determine whether the Sentry monitor can perform the actions configured in the Sentry profile or not.

6.2.7.1 Case 1: Endpoint Manager is Unavailable

The Endpoint Manager becomes unavailable when the Sentry monitor is monitoring something on the Endpoint. What happens? To find out, we performed the following tests.

- Logging to the Endpoint Manager
- Logging to the Endpoint gateway
- Logging to the Endpoint
- Sending a TEC event to the TEC server

As a result, logging to the Endpoint gateway and logging to the Endpoint were still functional even if the Endpoint Manager became unavailable (refer to Figure 94 on page 239). The next table summarizes the results.

Table 42. Sentry Monitor without Endpoint Manager

Monitor Action	Result	Auto Recover when EP Manager becomes available
Logging to EP Manager	Not Available	Yes
Logging to EP gateway	Available	-
Logging to Endpoint	Available	-
Sending T/EC Event	Not Available	Yes

In the above table, the "Auto Recover when EP Manager becomes available" field means the Sentry monitor actions will work correctly again without any operation when the Endpoint Manager becomes available.

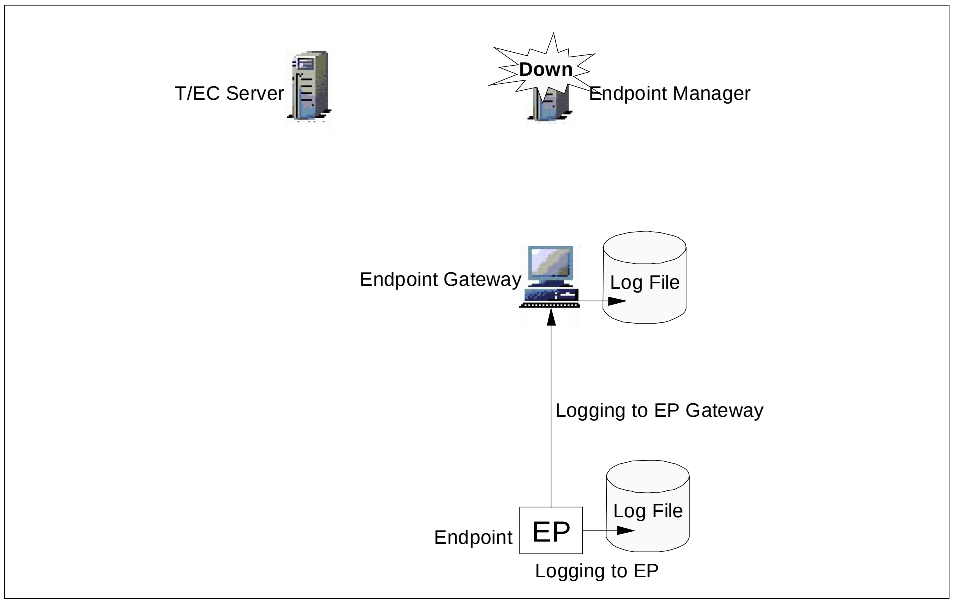


Figure 94. The Sentry Monitor without the Endpoint Manager

6.2.7.2 Case 2: Endpoint Gateway is Unavailable

In this case, we assume the Endpoint gateway becomes unavailable when the Sentry monitor is monitoring something on the Endpoint. What happens?

To discover this, we performed tests as in Case 1. We summarize the results as follows:

Table 43. The Sentry Monitor without Endpoint Gateway

Monitor Action	Result	Auto Recover when EP Gateway becomes available
Logging to EP Manager	Not Available	Yes
Logging to EP gateway	Not Available	Yes
Logging to Endpoint	Available	-
Sending T/EC Event	Not Available	Yes

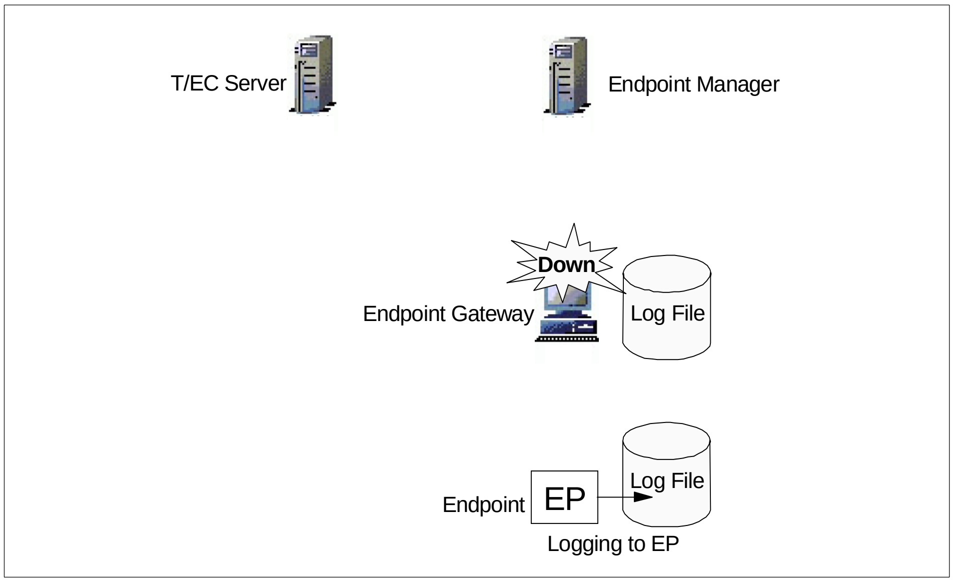


Figure 95. The Sentry Monitor without Endpoint Gateway

Note

In this case, we didn’t configure the alternate gateway. If there is any alternate gateway, what happens? You should hit upon the answer if you have already read chapter 5. When the Endpoint attempts to perform the Sentry monitor action and the Endpoint can’t perform it, the Endpoint recognizes it is isolated, so the Endpoint tries to connect to an alternate gateway. This means, if you configure the appropriate alternate gateway, the Endpoint recovers from this situation automatically. Of course, the Distributed Monitoring software must be installed to the alternate gateway before the Endpoint is isolated. But in this case, there is no alternate gateway; so nothing happens.

6.2.7.3 Case 3: Endpoint Gateway Migration

This case is a little different from the other cases. We assume the Endpoint gateway has migrated to the alternate gateway when the Sentry monitor is monitoring something on the Endpoint. What happens? To find out, we performed the tests as in case 1. We summarize the results as follows:

Table 44. Sentry Monitor and Endpoint Gateway Migration

Monitor Action	Result	Auto Recover when EP Gateway is Migrated to Original EP Gateway again
Logging to EP Manager	Available	Yes
Logging to EP Gateway	Available	Yes
Logging to Endpoint	Available	Yes
Sending T/EC Event	Available	Yes

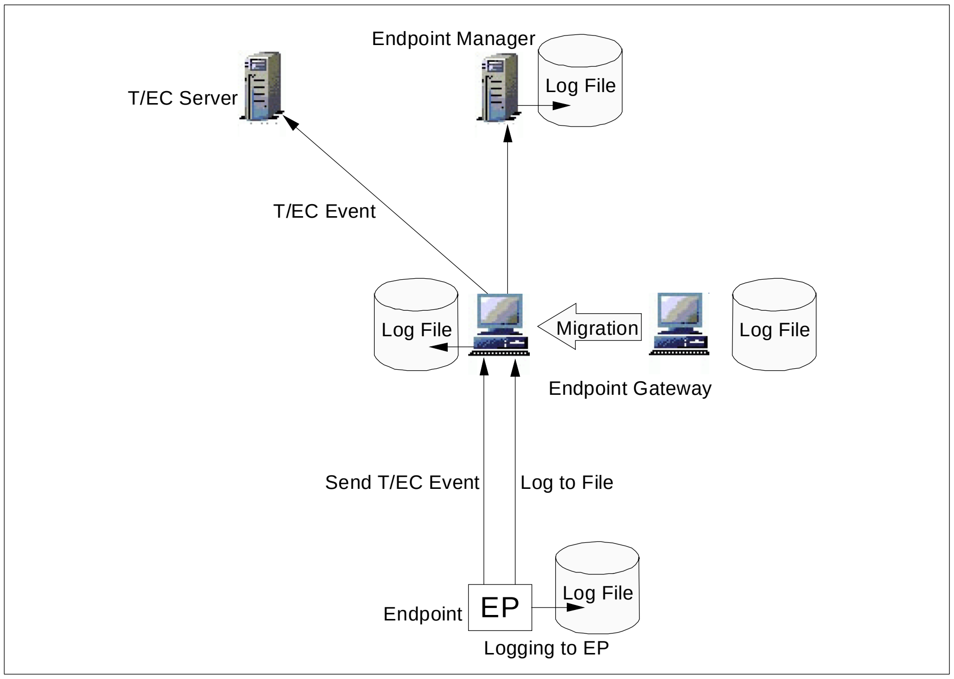


Figure 96. Sentry Monitor and Endpoint Gateway Migration

As you can see, the Endpoint configuration is very flexible and absolutely different from the configuration between the TMR server and Managed Node. Of course, Distributed Monitoring should be installed to the alternate gateway as well.

6.2.8 Database Backup and Sentry Monitors on the TMA

When you make a backup of the Tivoli object databases using the `wbkupdb` command, and some Sentry monitors are running on the Endpoints, can the `wbkupdb` store information about the Sentry monitors running on the Endpoints? The answer is no. Therefore, if you are forced to do a restore, all Sentry monitors running on Endpoints are forgotten. This means every time you restore the databases, you need to distribute all Sentry profiles to all the Endpoints if you would like to continue monitoring with the configurations you had when you made the backup.

Note

When you restore the databases using the `wbkupdb -r` command, you should reboot the Endpoint machine or kill the Sentry engine process (`dm_ep_engine.exe`) forcibly (not recommend this way), because the Sentry engine still continues monitoring with the latest configurations even if you restore the databases. To keep consistency, we recommend you to reboot the Endpoint machine after restoring the Tivoli object databases.

6.2.9 The wclreng Command and the TMA

The `wclreng` command is provided by Distributed Monitoring to clear the Distributed Monitoring engine. However, you should take care of this command when you use it for an Endpoint. Sometimes, the `wclreng` command disables the Sentry monitor function. Below, we describe the cases when the Sentry monitor becomes unavailable because of the `wclreng` command.

6.2.9.1 wclreng Command and Profile Distribution

In Version 3.6, we witnessed some unusual behavior that you should be aware of. After you clear the Sentry engine on the Endpoint, sometimes no Sentry profiles can be distributed. This situation may occur when you create and distribute the profile in a dataless profile manager. The following figure shows this situation.

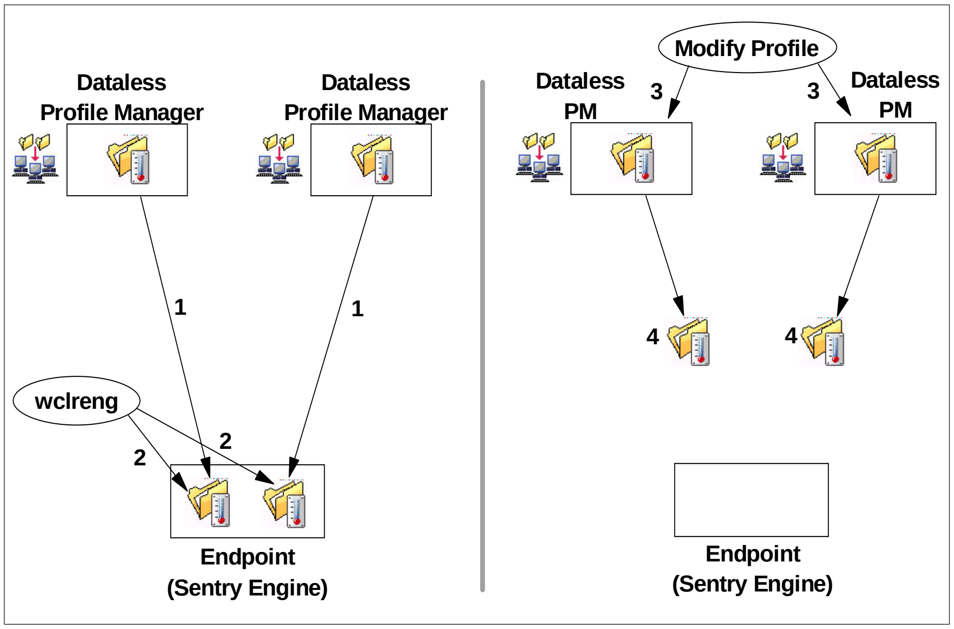


Figure 97. The wclreng Command and Profile Distribution

1. Distribute the Sentry profile defined in the dataless profile manager to the Endpoint and the monitor defined in the Sentry profile begins to monitor.
2. Then clear the Sentry engine using the `wclreng` command.
3. After clearing the engine, modify the Sentry profile in the dataless profile manager.
4. Then push the profile with an exact copy. After this operation, the profile seems to be distributed to the Endpoint without problem. But the Sentry monitor in the profile never monitors anything.

In this situation, there are no error messages in any log files; so it is difficult to make sense of what happened.

6.2.9.2 wclreng Command and Flush Engine

Here is another abnormal situation that arose in our testing. When you create the Sentry profile in the dataless profile manager and distribute it to the some Endpoints, be careful to clear the Sentry engine of the Endpoint using the `wclreng` command. Then the profile on all Endpoints will be cleared. The following figure shows this situation.

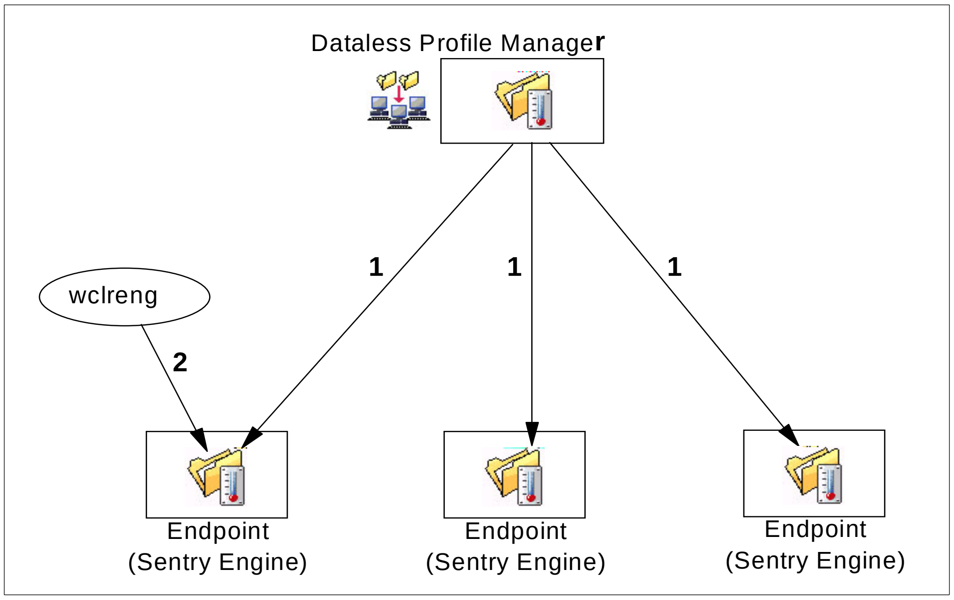


Figure 98. The wclreng Command and Flush Engine

1. Distribute the Sentry profile defined in the dataless profile manager to some Endpoints.
2. On one of the Endpoints, clear the Sentry engine using the `wclreng` command. Then the profiles distributed to the other Endpoints are deleted as well.

Note

As you can see, the `wclreng` used with a dataless profile manager may exhibit some problems. To avoid problems, you should keep in mind the following:

- It is best to create profiles in classic profile managers and use dataless profile managers as leaf nodes in your profile manager hierarchy. The dataless profile manager should be used as a container for Endpoint subscribers.
- Be careful when using the `wclreng` command in the above situation.

6.3 Downcall Applications and the TMA

As we mentioned, the downcall oriented applications are normally simple. Software Distribution is a typical downcall-oriented application and a good example for understanding downcall-oriented applications. In this section, we focus on Software Distribution and discuss the implementation considerations, how Software Distribution is implemented on the TMA, and the interaction between Software Distribution and the TMA.

6.3.1 Software Distribution and TMA

The deployment applications are also very popular applications for most customers, and Software Distribution is one of the most-used applications chosen from the Tivoli Management Applications suite.

File transfer and data transfer are the most typical subjects of systems management. Tivoli provides Software Distribution, and Version 3.6 of the Software Distribution supports the TMA. Basically, the operations are almost the same as in the previous version of Software Distribution, but there are some considerations for implementing it on the TMA. In this part of the book, we introduce the considerations, hints and tips.

6.3.2 Installation Consideration

Software Distribution needs a special module for the Endpoint gateway; so you have to make sure that you install the module to the gateways that the Endpoint will be logged in to. The module is shown in the installation panels as:

```
TME 10 Software Distribution Gateway, Version 3.6
```

If you forget to install the Software Distribution Gateway module to the Endpoint gateway and attempt to distribute any file package to the Endpoint, the following error messages will appear in the log file for Software Distribution.

```
File Package:  "Notes_ID"
Operation:     install  (m=5)
Finished:      Wed Sep 30 11:32:43 1998
--------
Source messages:
<none>
--------
ishii:FAILED
Wed Sep 30 11:32:43 CDT 1998 (4): resource '/usr/local/Tivoli/bin/
lcf_bundle//bin/w32-ix86/TAS/MANAGED_NODE/fp_endpoint' not found
==============
```

6.3.3 MDist Repeater and Endpoint Gateway

The MDist repeater function is one of the most important functions provided by the Tivoli Management Framework service because data and information can be transferred through the MDist repeater with the fan-out feature in the TMR.

The Endpoint gateways are configured as MDist repeaters automatically when you create them; so you don't need to do any configuration for the MDist repeater in the TMA environment. In other words, the Endpoint gateway has the MDist repeater capability built in. If you are currently using a Managed Node to repeat for MDist and you then create an Endpoint gateway on the Managed Node, that Endpoint gateway becomes the MDist repeater. You can no longer use the Managed Node as an MDist repeater.

At that time, the creating process of the Endpoint gateway removes the current MDist repeater definitions from the Managed Node; then the creating process creates the new MDist repeater on the Endpoint gateway. However, all parameters defined on the former MDist repeater are passed to the new MDist repeater; so your turning configurations for the former MDist repeater will be kept.

6.3.4 Software Distribution and Endpoint Method

Software Distribution provides file or data transfer function to any managed system in the Tivoli Management environment. All operations for Endpoints would use the downcall. To understand the interaction between the Endpoint and other managed resources, we introduce the example of the file package distribution shown in Figure 99. In this example, we are configured as follows (see Table 45).

Table 45. The Sample Configuration of Software Distribution

Configuration	Location
File Source Host	Endpoint Gateway (`trout`)
Destination Host	Endpoint (`ishii`)
Log File	Endpoint Manager (`ishii2`)
Sending Tivoli Notify	None

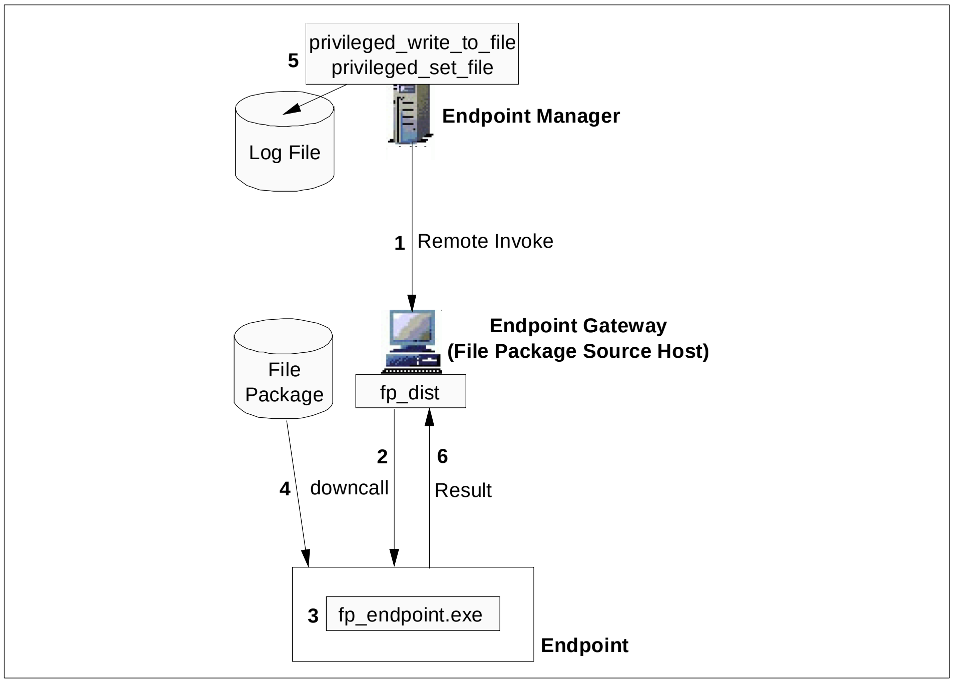


Figure 99. Distributing a File Package

1. When we executed the `wdistrib` command on the TMR server for distributing the file package to the Endpoint, the `fp_dist` method was invoked on the file package source host remotely.
2. The `fp_dist` method issued the downcall for starting the Endpoint method. Then the dependency methods defined in the dependency manager would be checked and downloaded if the Endpoint method cache did not already contain them (or the correct version).
3. The `lcfd` daemon received the downcall from the Endpoint gateway and invoked the Endpoint method (`fp_endpoint.exe`) locally.
4. Then the file package was transferred from the Endpoint gateway to the Endpoint.
5. After transferring the file package, the `privileged_write_to_file` and `privileged_set_file` methods were invoked on the TMR server for logging the messages to the log file defined in the file package profile.
6. Then the `fp_dist` method received the result (normal end) from the Endpoint. If you requested the sending of a Tivoli notice, the Tivoli notice would be sent to the notification server at this time.

6.3.5 Understanding Software Distribution Behavior with TMA

As you know, Software Distribution is a typical downcall-oriented application; so basically, all components of the TMA environment, Endpoint Manager, Endpoint gateway and Endpoint must be available when you distribute the file package to any Endpoints.

Therefore, we assume the Endpoint gateway is migrated to the alternate gateway before we distribute the file package to the Endpoint. To make sure of whether the file package is sent to the Endpoint without an error or not, we performed a test and show the summary of the result as follows.

Table 46. Distributing File Package with Migration

Action	Result	Auto Recover when EP Gateway is Migrated to Original EP Gateway again
Distribute File Package	Available	Yes

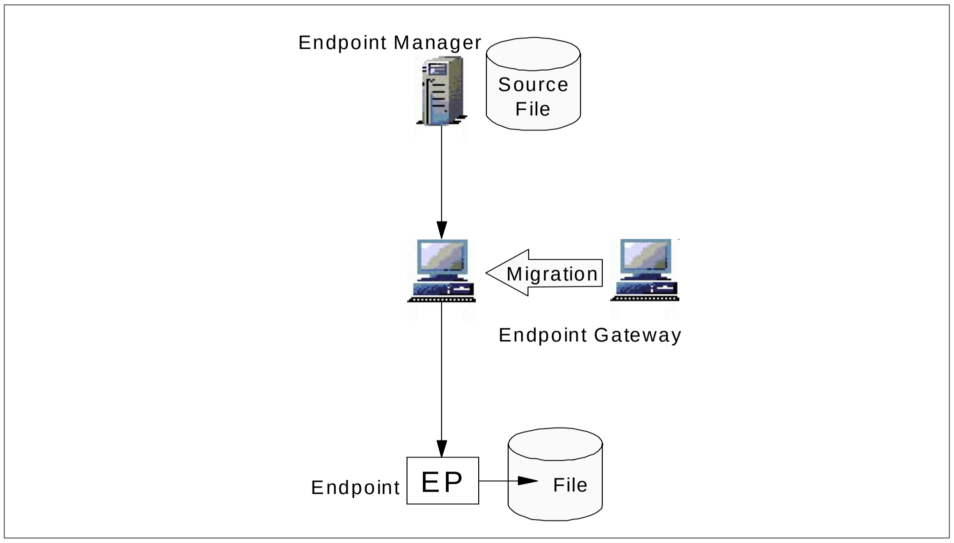


Figure 100. Distributing File Package with Migration

As you can see, the file package distribution was not affected by the Endpoint gateway migration. However, again, the Software Distribution gateway software must be installed on both of the Endpoint gateways in this case.

6.4 Other Applications and the TMA

Each Tivoli Management Application belongs to either the upcall-oriented application type or downcall oriented-application type because all applications must use the upcall or downcall to perform the management operations. Most of the Tivoli Management Applications are downcall-oriented applications. The following shows other typical applications for each type.

Upcall Application	Event Adapter
Downcall Application	User Administrator

Now you have an understanding of how the upcall applications work and how the downcall applications work. The detail of management operations are different among the applications; however, the overview of the behavior is similar. To understand it in detail, the log files (epmgrlog, gatelog, lcfd.log) and the `wtrace` should be useful, as in the examples we introduced. In the log files and in the `wtrace` output, the most important thing is what methods are being issued and whether they are upcalls or downcalls.

Chapter 7. Advanced Knowledge of the TMA

This chapter introduces advanced functions, as well as some undocumented features of the Tivoli Management Agent. These can prove helpful when using the TMA or when performing problem resolution.

7.1 The wadminep Command

The `wadminep` command has many uses. We have already seen how we can use it to force an upgrade of the `lcfd` daemon. However, the `wadminep` command provides not only the upgrade option, but also many powerful functions for managing the TMA. Note that many of the parameters we discuss are currently undocumented. This implies that some of these parameters may not be available in future releases of this command.

7.1.1 Normal Usage of the wadminep command

Usually, the `wadminep` command is used for upgrading `lcfd` process itself as follows.

```
# wadminep ishii upgrade
```

In above example, `ishii` is the label of the Endpoint. If the `wadminep` command detects that the version of the `lcfd` is lower than the available version at its Endpoint gateway, it upgrades the `lcfd` executable. Normally, this kind of upgrade process is executed as part of the Endpoint login_policy script.

7.1.2 Administrative Operations with the wadminep Command

The `wadminep` command performs many administrative operations on an Endpoint client. The `wadminep` command is run from a Managed Node.

7.1.2.1 Authorization

The administrative commands require the admin role for the administrator with the privilege of root or Administrator on the Endpoint. Browse commands require the user role.

7.1.2.2 Arguments

The `wadminep` command is run with the following syntax.

```
wadminep [-h] endpoint_label command [arguments]
```

The meaning of the each parameter is as follows:

`-h`	Display a detailed usage statement listing all the commands available on `wadminep`.
`endpoint_label`	Specifies the label of the Endpoint on which the command will run.
`command`	Specifies the operation to run on the specified endpoint.

7.1.2.3 Administrative Command

The following are the administrative commands we can specify with the `wadminep` command.

`copy_file old_filename new_filename`	Copy the old_filename to the new_filename.
`change_file_mode filename mode`	Change the permissions on the specified file. This command is valid on UNIX platforms only.
`create_directory dir_name`	Create the specified directory on the Endpoint.
`exec_process`	Execute a process on the Endpoint (blocking mode).
`get_file source_file destination_file`	Retrieve a file from the Endpoint and write its contents to the specified destination.
`kill_process`	Stop a process on the Endpoint.
`move_file old_filename new_filename`	Move the old_filename to the new_filename on the Endpoint.
`reboot_system`	Reboot the specified Endpoint machine. This command is valid on Windows platforms only.
`reexec_lcfd [config_arg]`	Stop and restart the Endpoint daemon (lcfd) process. Optionally, you can include arguments from the `lcfd` command to restart the Endpoint daemon with a different configuration.
`remove_file filename`	Remove the specified file from the Endpoint.

`rollback`	Revert the Endpoint back to the previous version of the Endpoint software.
`send_file source_file destination_file`	Send the specified file to the destination_file on the Endpoint.
`spawn_process process`	Spawn the specified process on the Endpoint (non-blocking mode).
`shutdown_lcfd endpoint_label`	Stop the Endpoint daemon process. Use this command with caution. Shutting down the Endpoint severs your remote connection to the Endpoint. You will have to manually restart the Endpoint.
`upgrade`	Upgrade the Endpoint with information stored in the gateway repository. Use this option to upgrade to newer versions of the Endpoint software.

7.1.2.4 Browse Command

The following are the browse commands we can specify with the `wadminep` command.

`view_bin_dir`	Return the location of the binary directory.
`view_cache_index`	Display the Endpoint cache index file.
`view_config_info`	Display the Endpoint configuration contained in the last.cfg file.
`view_directory pathname`	Display a file list for the specified directory.
`view_file filename`	Display the contents of the specified text file to the screen.
`view_interpreter`	Return the Endpoint interpreter type.
`view_lib_dir`	Return the location of the library directory.
`view_log_file`	Display the contents of the Endpoint log file.
`view_machine_id`	Display the Endpoint's unique machine ID.

`view_run_dir`	Return the location of the directory from which the Endpoint is running.
`view_stage_dir`	Return the location of the staging directory.
`view_statistics`	Display a list of Endpoint statistics including cache size, cache hits, downcall hits and miss, downcall history, and amount of time the Endpoint has been running.
`view_system_dir`	Return the location of the system directory.
`view_temp_dir`	Return the location of the temporary directory.
`view_upgrade_log`	Display the Endpoints of the upgrade history log.
`view_version`	Display the version of Endpoint code currently running on the Endpoint.

Note

Use these commands with caution. They are not officially supported by Version 3.6 of the Tivoli Management Framework. Only the `upgrade` option is officially supported.

7.2 TMA and the Tivoli Object Database

Basically, most of the administrative operations for the TMA can be done using the Desktop or CLI (`w` commands); however, understanding the Tivoli object database may help you with problem determination. In this section, we introduce hints and tips for understanding the TMA and the Tivoli object database.

7.2.1 Location of Tivoli Object Database

The Endpoint Manager uses both the TNR (Tivoli Name Registry) and its own private `.bdb` files to keep the track of the TMA data. To resolve the scale goals of the TMA architecture with the design of Tivoli's B-tree implementation, there is one `.bdb` file per Endpoint gateway. The following figure shows the structure of the Tivoli object database in the TMA environment.

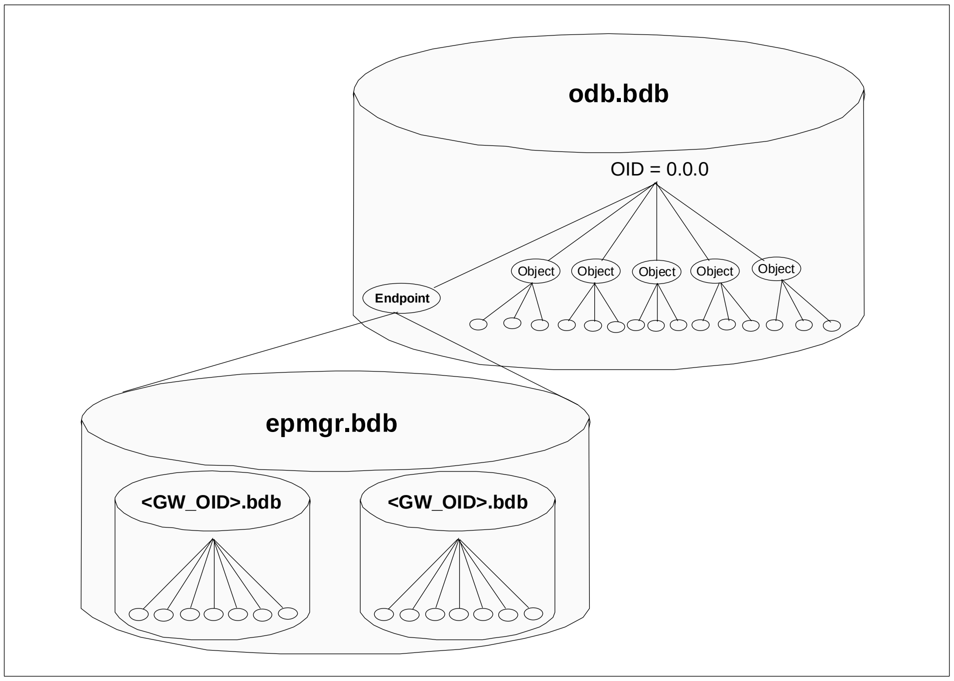


Figure 101. The epmgr.bdb Database

As you can see, this implementation allows us to manage thousands of Endpoints in a single TMR. The Tivoli object master database (odb.bdb file) does not contain detailed information regarding the Endpoints. The odb.bdb database just contains entries for each Endpoint. The detailed information for each Endpoint, for example the login_interfaces information, are actually contained in the epmgr.bdb database. The epmgr.bdb database actually contains the databases (.bdb files) for each Endpoint gateway which is managed by the Endpoint Manager. The name of this file is the object reference of the related Endpoint gateway. In addition, as expected, there is the name registry entry for each Endpoint. Attached to this entry is the object reference of the assigned gateway.

For example, the following is a sample of the <gw_oid>.bdb file in the Endpoint Manager.

```
panda:root(/) ls $DBDIR/epmgr.bdb
1956524575.1.530.bdb  1956524575.2.21.bdb
1956524575.1.530.log  1956524575.2.21.log
panda:root(/)
```

In this example, 1956524575.1.530 and 1956524575.2.21 are the object references of the Endpoint gateways which are managed by the Endpoint Manager. The following figure shows the physical location of each Tivoli object database for managing the TMA.

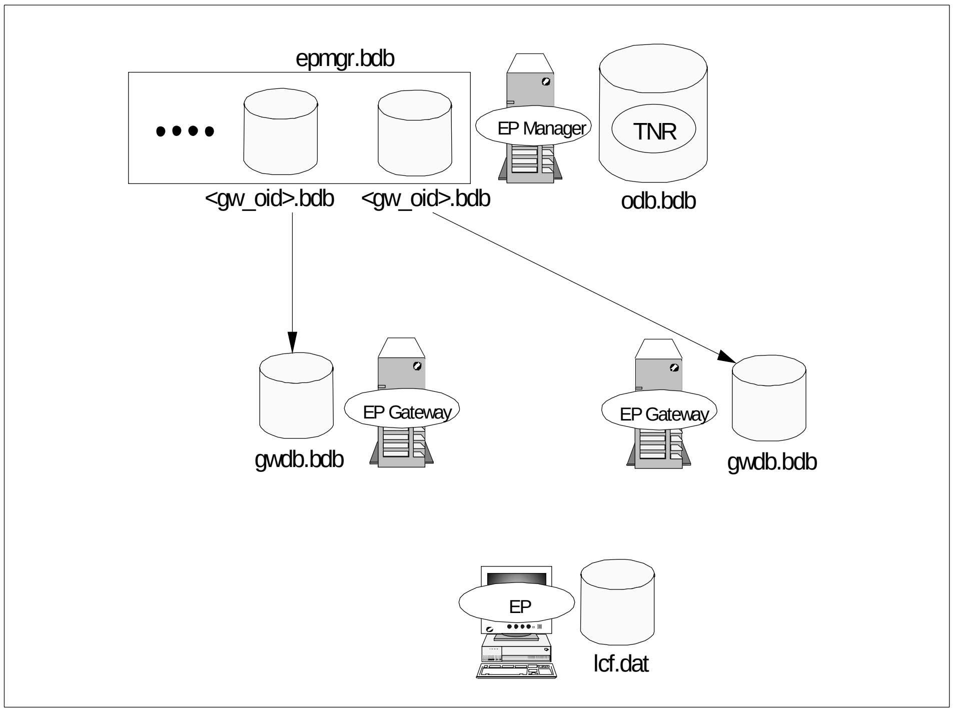


Figure 102. Tivoli Databases for Managing TMA

As you can see, the Endpoint gateways have a copy of the <gw_oid>.bdb database for each Endpoint gateway as a cache. The name of the database file in the Endpoint gateway is the gwdb.bdb file. The gwdb.bdb database contains not only a copy of the <gw_oid>.bdb database of the Endpoint Manager but also other information for the Endpoints which have logged into the Endpoint gateway. The gwdb.bdb database is synchronized with the <gw_oid>.bdb database each time the Endpoint gateway boots, and you can use the wep sync_gateways command for synchronizing the Endpoint information stored by the Endpoint Manager, Endpoint gateway and Endpoint (lcf.dat file) manually.

The Endpoint has a data file for Endpoint login. Actually, this is not a Tivoli object database, but contains important information for the Endpoint behavior. The following table shows the actual location of each database.

Table 47. Tivoli Database Locations

Database	Location	Directory
odb.bdb	EP Manager	$DBDIR/odb.bdb
gw_oid.bdb		$DBDIR/epmgr.bdb/<gw_oid>.bdb
gwdb.bdb	EP gateway	$DBDIR/gwdb.bdb
lcf.dat	Endpoint	C:\Tivoli\lcf\dat\1\lcf.dat

In this table, $DBDIR is the /var/spool/Tivoli/<servername>.db directory by default.

7.2.2 Contents of TMA Databases

To manage a TMA, each database is created, modified and managed by each managing system, the Endpoint Manager, Endpoint gateway and Endpoint. The following figure shows an overview of the contents of these databases.

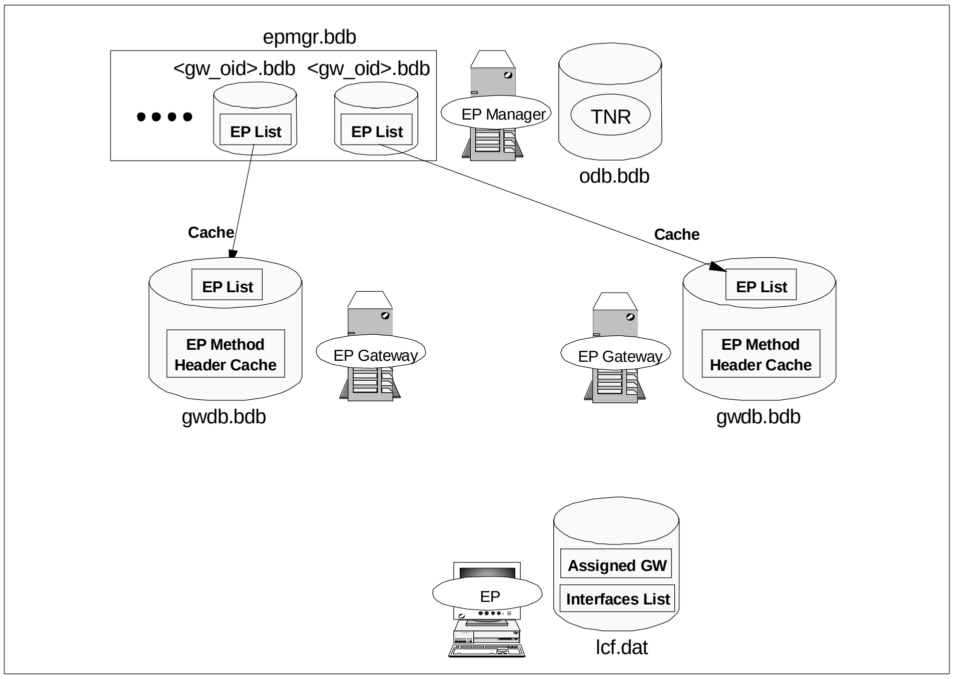


Figure 103. The Contents of Tivoli Object Databases for Managing TMA

odb.bdb This database on the TMR Server contains a lot of information for managing the whole TMR, and is considered the master database of the TMR. Most of the data are for managing full Managed Nodes or PC Agents. This database contains the Tivoli Name Registry (TNR). The TNR is a quick lookup table for object labels and object IDs.

The TNR contains the Endpoint label, Endpoint object ID and last known Endpoint gateway object ID. A process (for instance, the `wep` command) can use this information in the TNR to look up the assigned gateway by the Endpoint label. Of course, the label of the Endpoint gateways and Endpoint Manager are included in the TNR as well. There are also odb.bdb databases on each Managed Node, but they do not contain the TNR.

<gw_oid>.bdb This database on the TMR Server contains the Endpoint list for the Endpoint gateway. The Endpoint Manager is responsible for managing the Endpoint list and Endpoint gateway mapping for all Endpoints and Endpoint gateways in the TMR. Therefore, the Endpoint list is updated by the Endpoint Manager every time an Endpoint is added or deleted. The Endpoint list contains the information about which Endpoints are logging into the Endpoint gateway and the Endpoint's information. This database is located in the $DBDIR/epmgr.bdb/<gw_oid>.bdb on the Endpoint Manager.

gwdb.bdb Each Endpoint gateway has a cache of the <gw_oid>.bdb database that contains only the Endpoints that the Endpoint gateway is responsible for. This database consists of two major parts. One is, as we mentioned, a cache of the Endpoint list which is managed by the Endpoint Manager. The Endpoint list is read in to the cache from the Endpoint Manager when the Endpoint gateway starts. Another is the information about the Endpoint methods. This is the cache of the imdb.bdb database of the Endpoint Manager, and it contains the method header information. This database is located in the $DBDIR/gwdb.bdb on the Endpoint gateways.

lcf.dat This data file is actually not a Tivoli object database. However, this data file is very important for the Endpoint configuration. The assigned gateway information and the Endpoint login interfaces list are contained in this data file. The information included in the lcf.dat file is equivalent to the login_info structure defined in the mrt_wire_adr.h header file. If you install Tivoli ADE, you can find this header file under the C:\Tivoli\bin\lcf_bundle\include\w32-ix86\mrt directory. We developed a special program that shows the contents of the lcf.dat file as follows.

```
C:\Tivoli\lcf\dat\1>dumpdat
version=131077
status=1
hostname=salmon
interp=w32-ix86
unique_id=1QGF6G16V4+9Y5YTPXBH0000054E
httpd_password=tivoli:{jgU[-rn
odnum=291
region=1189622596
ep addr=9.3.1.193+1029
gw addr=9.3.1.133+9494
gw_alias addr=9.3.1.133+9494
login i/f addr=9.3.1.133+9494
login i/f addr=9.3.1.134+9494
login i/f addr=9.3.1.149+9494
login i/f addr=9.3.1.133+9494
C:\Tivoli\lcf\dat\1>
```

7.2.3 The wbkupdb Command and TMA Information

From the previous section, it should be clear that with the TMA architecture, new considerations arise with respect to the use of the `wbkupdb` command.

The `wbkupdb` command will back up and restore both the name registry and the Endpoint Manager `.bdb` files. This is consistent with the historical semantics of the `wbkupdb` command. However, it also creates an issue with TMA management. If you perform a restore, how does that affect the Endpoints that have logged in since the last backup? The answer is that they are all forgotten. While not ideal, this is the current behavior of Tivoli 3.6. In order to work around this, one must manage the TMA data by hand with respect to the `wbkupdb` command.

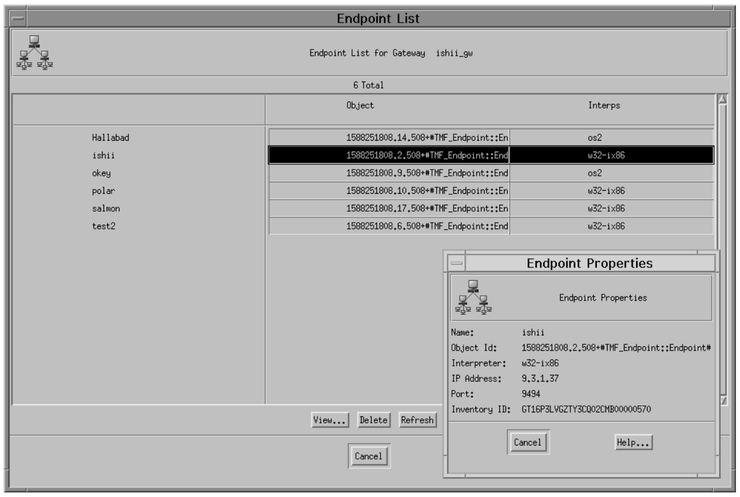
To restore the Endpoint information correctly, when you restore the backup, you need to restart the `oserv` daemon (`odadmin reexec`) of the TMR server. If you don't do that, the `wep ep_name status` command returns the unreachable status even if you can find the `ep_name` in the output of the `wep ls` command.

7.2.4 Exploring the Tivoli Object Database

The Tivoli object database stores object information, and each object that is stored in the database has a unique object ID (OID). The format of the Tivoli object database is proprietary; therefore, we use one of the following ways to access the entries related to the TMA in the Tivoli object database.

- Use the Desktop interface
- Use the `w` commands
- Use low-level commands

The Desktop interface provides very limited function for browsing the TMA entries of the database. We are able to delete a TMA entry from the database using Desktop (refer to Figure 104).



	Object	Interps
Hallabad	1588251808.14.508+#TMF_Endpoint::En	os2
ishii	1588251808.2.508+#TMF_Endpoint::End	w32-ix86
okey	1588251808.9.508+#TMF_Endpoint::End	os2
polar	1588251808.10.508+#TMF_Endpoint::En	w32-ix86
salmon	1588251808.17.508+#TMF_Endpoint::En	w32-ix86
test2	1588251808.6.508+#TMF_Endpoint::End	w32-ix86

```
Name:          ishii
Object Id:     1588251808.2.508+#TMF_Endpoint::Endpoint#
Interpreter:   w32-ix86
IP Address:    9.3.1.37
Port:          9494
Inventory ID:  GT16P3LVGZTY3CQ02CMB00000570
```

Figure 104. The Endpoint List after the Endpoint Login

Most of the administrative operations regarding the TMA can be performed using the `w` commands. The following `w` commands are used for modifying or deleting the TMA entries in the database. The `wep` command is one of the most useful and frequently used `w` commands for TMA operations.

In some cases, the `w` commands may not provide the level of access we need. In this case, knowledge of the low-level commands is very useful and should help you.

7.2.4.1 Exploring Endpoint Gateway Entries

The resource type `Gateway` was added for implementing the LCF architecture and for supporting the TMA. We can check it easily by using the `wlookup -R` command. Browsing or modifying the entries under the `Gateway` resource type (object) is not so difficult because it is the same as for other resource types, such as a `ManagedNode`. The following example shows how to change the label of the Endpoint gateway.

1. First of all, we need to know the object ID (OID) of the Endpoint gateway.

```
# wlookup -ar Gateway
ishii_gw 1588251808.1.819#TMF_Gateway::Gateway#
```

2. Then we execute the `objcall` command for browsing all the entries under the gateway's OID.

```
# objcall 1588251808.1.819 contents
ATTRIBUTE:_BOA_id
ATTRIBUTE:allow_logins
ATTRIBUTE:class_objid
ATTRIBUTE:collections
ATTRIBUTE:debug_level
ATTRIBUTE:flags
ATTRIBUTE:impl_root_a
ATTRIBUTE:interps
ATTRIBUTE:label
ATTRIBUTE:max_concurrent_jobs
ATTRIBUTE:port
ATTRIBUTE:pres_object
ATTRIBUTE:pro
ATTRIBUTE:pro_name
ATTRIBUTE:protos
ATTRIBUTE:proxy
ATTRIBUTE:resource_host
ATTRIBUTE:run_policy
ATTRIBUTE:session_timeout
ATTRIBUTE:skeleton
ATTRIBUTE:sort_name
ATTRIBUTE:state
ATTRIBUTE:the_path
```

3. Now we can get these attributes of the gateway using the `idlattr` command.

```
# idlattr -t -g 1588251808.1.819 label string
"ishii_gw"
```

4. To change the label of the gateway, we run the `idlattr` command as follows:

```
# idlattr -t -s 1588251808.1.819 label string '"new_label"'
```

As you can see, we can explore the Endpoint gateway object hierarchy using the objcall or idlattr command.

7.2.4.2 Exploring Endpoint Manager's Entries

Exploring the entries under the Endpoint Manager object is the same as the case of the Endpoint gateway object. In this example, we introduce how to change the max_jobs attribute of the EndpointManager object. The max_jobs attribute is a value for controlling how many tasks the Endpoint Manager can handle at the same time. Therefore, in large environments, this value may affect the throughput for Endpoint logins in the TMR. The default value of the max_jobs attribute is 20.

1. First of all, we need to know the Endpoint Manager's object ID (OID). The wlookup command is our command line interface into the Tivoli Name Registry (TNR) and can be used for searching the OID of the Endpoint Manager.

```
# wlookup -ar EndpointManager
epmgr_1588251808.1.517  1588251808.1.517
```

2. For browsing the attributes and methods of the EndpointManager object, we use the objcall command as follows:

```
# objcall 1588251808.1.517 contents
ATTRIBUTE:_BOA_id
ATTRIBUTE:actions
ATTRIBUTE:add_targets
ATTRIBUTE:after_install_policy
ATTRIBUTE:allow_install_policy
ATTRIBUTE:behavior
ATTRIBUTE:class_objid
ATTRIBUTE:class_type
ATTRIBUTE:collections
ATTRIBUTE:credentials_lookup
ATTRIBUTE:def_policies
ATTRIBUTE:delete_targets
ATTRIBUTE:dialog
ATTRIBUTE:extension
ATTRIBUTE:filters
ATTRIBUTE:impl_name
ATTRIBUTE:impl_type
ATTRIBUTE:indirect
ATTRIBUTE:initialized
ATTRIBUTE:interfaces
ATTRIBUTE:label
ATTRIBUTE:max_iom_records
ATTRIBUTE:max_jobs
ATTRIBUTE:members
ATTRIBUTE:migrate_targets
ATTRIBUTE:pres_object
ATTRIBUTE:pro
ATTRIBUTE:pro_name
ATTRIBUTE:prototypes
ATTRIBUTE:resource_host
```

```
ATTRIBUTE:run_policy
ATTRIBUTE:select_gateway_policy
ATTRIBUTE:skeleton
ATTRIBUTE:sort_name
ATTRIBUTE:state
ATTRIBUTE:val_policies
ATTRIBUTE:version
METHOD:_get_add_targets
METHOD:_get_credentials_lookup
METHOD:_get_delete_targets
METHOD:_get_max_iom_records
METHOD:_get_max_jobs
METHOD:_get_migrate_targets
METHOD:_get_run_policy
METHOD:_set_add_targets
METHOD:_set_credentials_lookup
METHOD:_set_delete_targets
METHOD:_set_max_iom_records
METHOD:_set_max_jobs
METHOD:_set_migrate_targets
METHOD:_set_run_policy
METHOD:add_boot_method
METHOD:add_gateway
METHOD:after_install_policy
METHOD:allow_install_policy
METHOD:batch_get_cache_info
METHOD:cache_dump
METHOD:create_gateway
METHOD:delete_endpoints
METHOD:delete_gateway_info
METHOD:delete_gateways
METHOD:display_view
METHOD:endpoint_add_backref_optimized
METHOD:endpoint_login
METHOD:endpoint_move_to_policy_region
METHOD:endpoint_remove_backref
METHOD:endpoint_set_label
METHOD:endpoint_view
METHOD:epmgr_boot
METHOD:fixup_tnr
METHOD:gateway_info_dump
METHOD:gateway_is_booting
METHOD:get_endpoint_cache
METHOD:get_endpoint_key_value
METHOD:get_endpoints
METHOD:get_endpoints_by_label
METHOD:get_gateway_info
METHOD:get_number_endpoints
METHOD:get_rpt_format
METHOD:goodbye
METHOD:has_endpoints
METHOD:iom_cache_dump
METHOD:iom_get_endpoint_cache
METHOD:list_boot_methods
METHOD:migrate
METHOD:orphan_gateway
METHOD:register_callback
METHOD:remove_boot_method
METHOD:remove_callback
METHOD:select_gateway_policy
METHOD:set_endpoint_key_value
METHOD:set_gateway_info
METHOD:set_last_gateway
```

```
METHOD:sync_gw_epcache
METHOD:test_boot_method
METHOD:update_endpoints
METHOD:update_map
```

3. For getting the value of the attributes we can use the `idlattr` command. In this case we retrieve the `max_jobs` value as follows.

```
# idlattr -t -g 1588251808.1.517 max_jobs short
20
```

4. Now, we modify this value from 20 to 30 using the `idlattr` command.

```
# idlattr -t -s 1588251808.1.517 max_jobs short 30
```

Note

The `max_jobs` attribute is not supported by Version 3.6.1 of the Tivoli Management Framework. Instead of `max_jobs`, other new attributes are supported by Version 3.6.1 of the Tivoli Management Framework. Please refer to the Chapter 10, “Tivoli Management Agent Performance Conisderations” on page 385 for more information about this subject.

7.2.4.3 Exploring Endpoint Entries

As we mentioned, the information on the Endpoint is not stored in the odb.bdb object database; so this case is a little different from the other cases with the Endpoint gateway and Endpoint Manager.

When the Endpoint Manager needs to map the Endpoint object reference or the label to its record in the Endpoint Manager .bdb file, it first performs the name registry lookup to find the assigned gateway. Then it performs a normal key lookup in the indicated .bdb file. After that, the related cache is updated if it is needed. The following figure shows this operation.

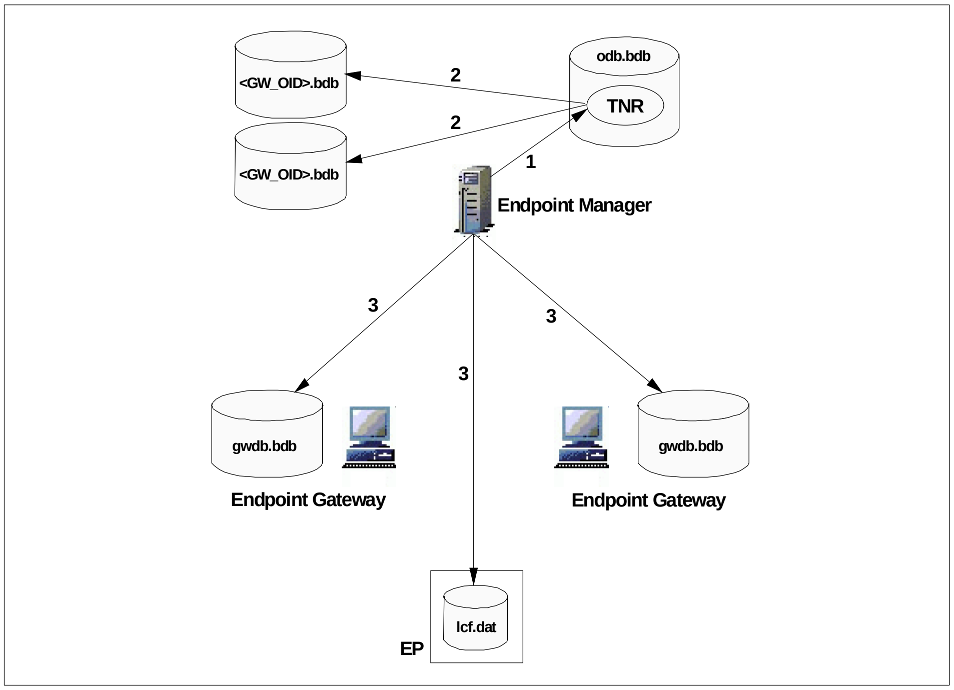


Figure 105. Accessing the Endpoint Entry in the Tivoli Object Databases

1. In this case, we assume a migration operation. First of all, the Endpoint Manager performs a TNR lookup to obtain the object reference of the Endpoint and its assigned gateway, which are related to each other during the migration by the Endpoint label as follows:

   ```
   wlookup -r Endpoint ep_label
   ```

   and

   ```
   idlcall $TMR.1.26 lookup '"Endpoint"' '"ep_label"'
   ```

2. Then the Endpoint Manager updates the indicated <gw_oid>.bdb database.

3. To keep consistency of the databases, the Endpoint Manager updates the appropriate cache (`gwdb.bdb`) as well. In this process, if the Endpoint Manager needs to update the other databases, for example the `lcf.dat` or TNR, the Endpoint Manager updates these databases as well. For instance, if you execute the `wep set gateway` or `wep set interfaces` command, the Endpoint Manager must update the `lcf.dat` file.

Accessing Endpoint information is handled somewhat differently. To browse the Endpoint information stored in the epmgr.bdb database, the Tivoli

Management Framework provides special commands, such as the `wep` command. These commands are available only for the Endpoint or Endpoint gateway gateways. In other words, most of the `w` commands do not interact directly with Endpoints (such as `wping`).

Since the Endpoint information is not stored in the odb.bdb database, we cannot access the data under the Endpoint object using the normal `objcall` or `idlattr` commands, which were introduced in the previous section. Therefore, if you would like to access the Endpoint information, you need to use one of the following methods.

- Using the special `w` commend. (such as, `wep`)
- Invoke the method which accesses the Endpoint information using the `idlcall` command.

For example, when you change the label of the Endpoint, you can use the `idlcall` command to invoke the `_set_label` method as follows:

```
idlcall -T top 1588251808.17.508+ _set_label '"ep_label"'
```

where the `1588251808.17.508+` is the object ID of the Endpoint which will be changed in the label, and the `ep_label` is the desired label. To change the Endpoint label, this is the only way because there is no `w` command to do that. The information about the Endpoint label is stored in both the TNR and Endpoint list (epmgr.bdb). In this case, the information is updated by the `_set_label` method.

7.3 Boot Process

To support the TMA, processes are run on each of the managing resources. In this section, we introduce the processes running on the Endpoint Manager, Endpoint gateway, and Endpoint, and how these processes are started. The following figure shows how each process starts automatically.

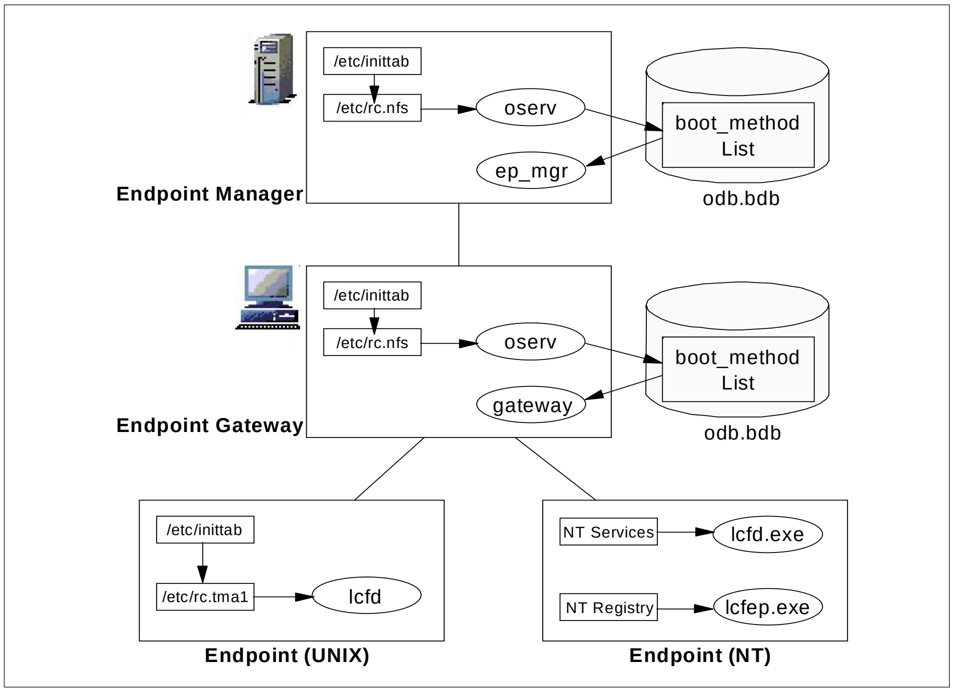


Figure 106. Boot Method of Each Process

7.3.1 ep_mgr Process

The ep_mgr process runs on the TMR Server and plays the role of Endpoint Manager. This process is executed by the oserv daemon automatically using its boot_method list when the oserv daemon starts. The oserv daemon is executed automatically from the definition in the /etc/inittab file when the system boots. The boot_method list contains the definition of the process, which is started automatically by the oserv daemon when the oserv daemon starts. To confirm this, you can check the boot_method definition as follows:

```
# objcall 0.0.0 self
1956524575.1.0
# objcall 1956524575.1.0 getattr oserv
1956524575.1.2
# objcall 1956524575.1.2 boot_method list
Scheduler
EndpointManager
HTTPd
ActiveDesktopList
#
```

As you can see, the `EndpointManager` is defined in the boot_method list of the Endpoint Manager machine. Therefore, the Endpoint Manager process (ep_mgr) is executed automatically every time the `oserv` daemon starts.

The boot_method list is configurable; so you can modify the entry in the boot_method list using the `objcall` command if you need to.

7.3.2 The gateway Process

The gateway process runs on the Endpoint gateway and plays the role of the Endpoint gateway. This process is also executed by the `oserv` daemon automatically using its boot_method list when the `oserv` daemon starts. The following is the sample of the boot_method list on the Endpoint gateway machine.

```
# objcall 0.0.0 self
1956524575.2.0
# objcall 1895624683.2.0 getattr oserv
1956524575.2.2
# objcall 1895624683.2.2 boot_method list
HTTPd
EventServer
SentryEngine
1956524575.2.41#TMF_Gateway::Gateway#
#
```

The `1956524575.2.41#TMF_Gateway::Gateway#` entry is the definition to start the Endpoint Gateway process (gateway) automatically.

7.3.3 The lcfd Process for the UNIX Endpoint

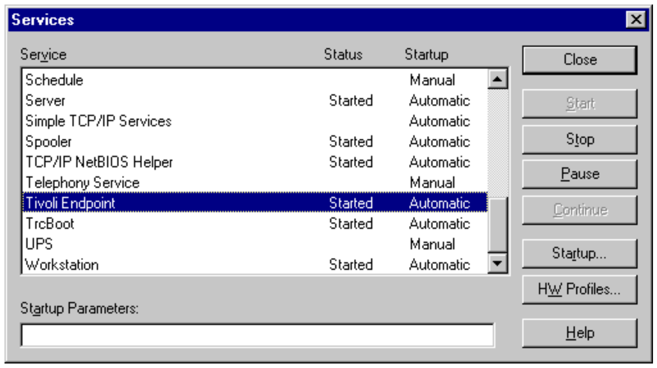
The `lcfd` process runs on the UNIX Endpoint and plays the role of the Endpoint. This process is executed automatically from the definition in the /etc/inittab file when the system boots.

When the Endpoint software is installed, the installation process creates an entry in the /etc/inittab file and /etc/rc.tma1 file to start the `lcfd` process every time the system boots.

7.3.4 lcfd.exe Process for NT Endpoint

The `lcfd.exe` process runs on NT and plays the role of the Endpoint. This process is executed automatically by the NT Services definition.

When the Endpoint software is installed, the installation process defines the NT service for the Endpoint (Tivoli Endpoint) to start the lcfd.exe process every time the system boots. Then the `lcfd` daemon starts automatically when the system boots. The following figure shows the configuration dialog for the NT services. In this figure, the Tivoli Endpoint service is configured as being automatic.



Service	Status	Startup
Schedule		Manual
Server	Started	Automatic
Simple TCP/IP Services		Automatic
Spooler	Started	Automatic
TCP/IP NetBIOS Helper	Started	Automatic
Telephony Service		Manual
Tivoli Endpoint	Started	Automatic
TrcBoot	Started	Automatic
UPS		Manual
Workstation	Started	Automatic

Figure 107. NT Service for Endpoint

7.3.4.1 lcfep.exe Process

On a Windows Endpoint, the lcfep.exe process is running as well. This process is executed automatically by the following definition in the Windows registry.

```
lcfep:REG_SZ:"C:\Tivoli\lcf\bin\w32-ix86\mrt\lcfep.exe"
```

You can see the above definition in the following directory of the Windows registry editor.

```
\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

The `lcfep.exe` process displays the Tivoli logo mark when you log into the Windows system and displays the Tivoli icon in the Windows taskbar as well. The following figure shows the Tivoli logo mark and icon.

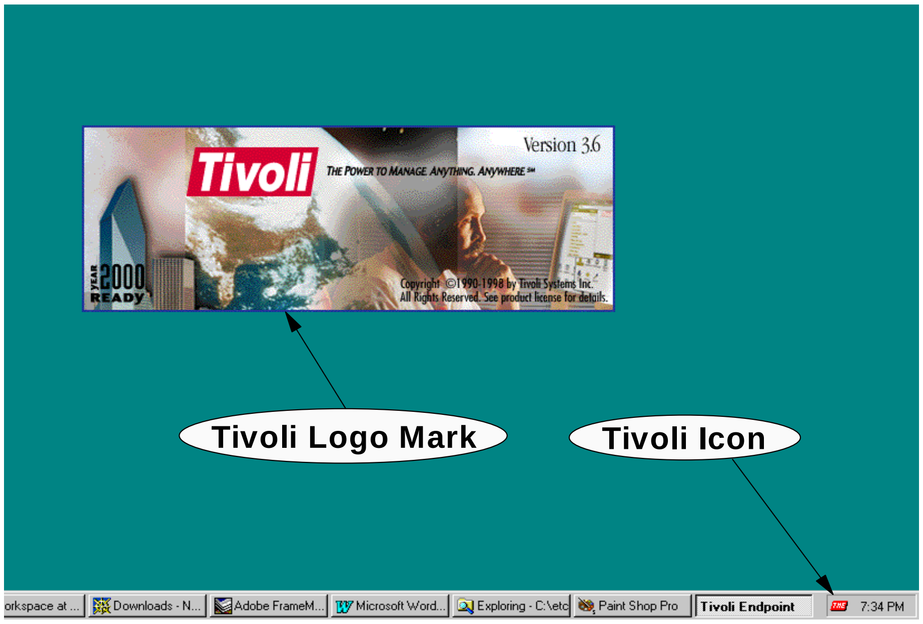
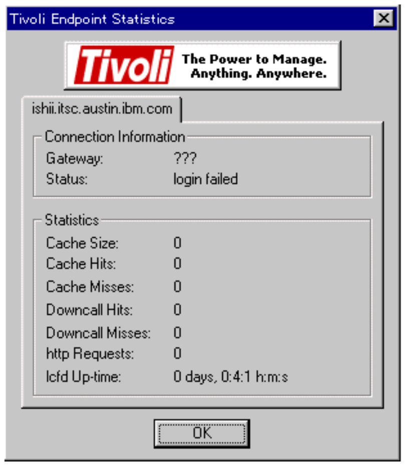


Figure 108. Tivoli Icon and Logo Mark

This process is independent of the lcfd.exe process; so the lcfep.exe starts and displays the icon even if the lcfd.exe process does not run. So, the existence of the Tivoli icon is a true indicator of the status of the lcf daemon itself.

This process can tell display Tivoli Endpoint statistics. To view the Endpoint statistics, click the Tivoli icon with the right button of the mouse; then select **View Statistics** (refer to Figure 109 on page 271). In this operation, the lcfep.exe process uses a pipe interface to get the information from the `lcfd` daemon.


Tivoli Endpoint Statistics
Tivoli The Power to Manage. Anything. Anywhere.
ishii.itsc.austin.ibm.com
OK

Connection Information

```
Gateway:        ???
Status:         login failed
```

Statistics

```
Cache Size:       0
Cache Hits:       0
Cache Misses:     0
Downcall Hits:    0
Downcall Misses:  0
http Requests:    0
lcfd Up-time:     0 days, 0:4:1 h:m:s
```

Figure 109. Tivoli Endpoint Statistic Window

7.4 Endpoint Login and Methods

During the Endpoint login process, some methods are invoked when the Endpoint login completes. In Chapter 5, “Anatomy of TMA Behavior” on page 135, we introduced how the Endpoint takes part in the TMR and how the Endpoint Manager and Endpoint gateway participate. Basically, each process corresponds to a method; so understanding which method is invoked during the Endpoint login helps you to know the internal operations of the TMA environment. In this section, we introduce the methods that are invoked during the Endpoint initial login process. The following figure shows the methods that are invoked in the Endpoint initial login process.

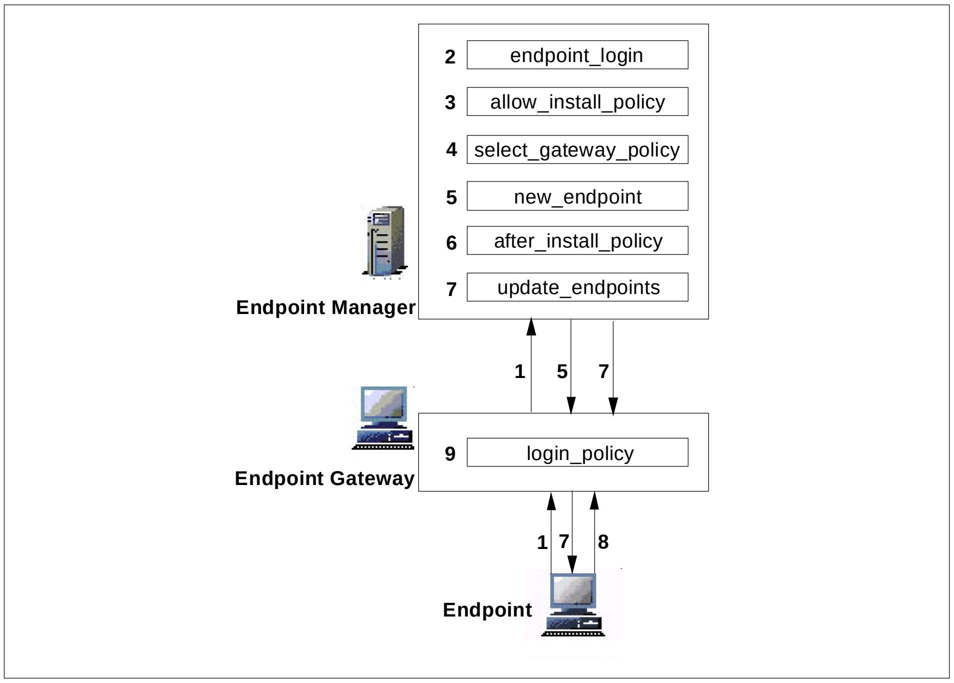


Figure 110. Endpoint Initial Login and Methods

1. The Endpoint sends the initial login request to the appropriate Endpoint gateway and the Endpoint gateway forwards the login request to the Endpoint Manager.
2. The Endpoint Manager recognizes that the initial login request arrives and starts the initial login process by issuing the `endpoint_login` method as follows.

```
loc-ic    53 M-hdoq     2-44      408
      Time run:    [Wed 02-Dec 11:45:58]

      Object ID:   1189622596.1.517#TMF_LCF::EpMgr#
      Method:      endpoint_login
      Principal:   root@panda.itsc.austin.ibm.com (-2/-2)
      Path:        __epmgr_implid
      Input Data: (encoded):

             {
               131077 0 "salmon" "w32-ix86" "FWQ7P94KRVTR0T02HHS300000577"
               0
               {
                 0
               }
               "NULL" 0 0
               {
                 0 1 1
                 {
                   16 "0x02 0x02 0x25 0x16 0x09 0x03 0x01 0xc1 0x00 0x00 0x00
                   0x00 0x00 0x00 0x00 0x00 "
                 }

               }

               {
                 0 0 0
                 {
                   0
                 }

               }

               {
                 0
               }

               {
                 2
                 [

                   {
                     0 1 1
                     {
                       16 "0x02 0x00 0x25 0x16 0x09 0x03 0x01 0x85 0x00 0x00
                       0x00 0x00 0x00 0x00 0x00 0x00 "
                     }

                   }

                   {
                     0 1 1
                     {
                       16 "0x02 0x00 0x25 0x16 0x09 0x03 0x01 0x95 0x00 0x00
                       0x00 0x00 0x00 0x00 0x00 0x00 "
                     }

                   }

                 ]

               }

             }
              "1189622596.2.19#TMF_Gateway::Gateway#"
```

3. Then the Endpoint Manager runs the `allow_install_policy` by issuing the `allow_install_policy` method.

```
loc-ic    54    M-H     1-53      252
      Time run:    [Wed 02-Dec 11:45:58]

      Object ID:   1189622596.1.517#TMF_LCF::EpMgr#
      Method:      allow_install_policy
      Principal:   root@panda.itsc.austin.ibm.com (-2/-2)
      Path:        /var/spool/Tivoli/panda.db/methghqJya
      Input Data: (encoded):

             {
               8
               [
                 "salmon" "OBJECT_NIL" "w32-ix86" "1189622596.2.19#TMF_Gatew
                 ay::Gateway#" "9.3.1.193+9494" "1189622596" "0" "5"
               ]

             }

             {
               1
               [
                 "LCF_LOGIN_STATUS=0"
               ]

             }

             {
               0
             }
```

4. After checking the return value of the `allow_install_policy`, the Endpoint Manager runs the `select_gateway_policy` by issuing the `select_gateway_policy` method. In this policy, to find the Endpoint gateway, some lookup operations are invoked.

```
loc-ic    55    M-H     1-53      252
      Time run:    [Wed 02-Dec 11:46:01]

      Object ID:   1189622596.1.517#TMF_LCF::EpMgr#
      Method:      select_gateway_policy
      Principal:   root@panda.itsc.austin.ibm.com (-2/-2)
      Path:        /var/spool/Tivoli/panda.db/methjZqJyb
      Input Data: (encoded):

             {
               8
               [
                 "salmon" "OBJECT_NIL" "w32-ix86" "1189622596.2.19#TMF_Gatew
                 ay::Gateway#" "9.3.1.193+9494" "1189622596" "0" "5"
               ]

             }

             {
               1
               [
                 "LCF_LOGIN_STATUS=0"
               ]

             }

             {
               0
             }
```

5. After referring to the select_gateway_policy, the Endpoint Manager attempts to connect to the Endpoint gateway by issuing the `new_endpoint` method. At that time, the epmgr.bdb database is updated if the Endpoint Manager can connect to the Endpoint gateway, and the Endpoint Manager assigns the dispatcher number and creates a private key for the Endpoint.

```
rem-ic    58    M-H     1-50      715
      Time run:    [Sun 31-Jan 17:38:34]

      Object ID:   1956524575.2.21#TMF_Gateway::Gateway#
      Method:      new_endpoint
      Principal:   root@panda.itsc.austin.ibm.com (0/0)
      Path:        __gateway_internals_implid
      Input Data: (encoded):
             "1956524575.5.508+#TMF_Endpoint::Endpoint#"  "ishii"
             {
               "C0F7NLVTS1CBVXH4CZDK00000593"
               {
                 0
               }
               "OBJECT_NIL" "OBJECT_NIL" "OBJECT_NIL" 0
               {
                 15 "0x15 0x5e 0x47 0xcd 0x31 0x92 0x1c 0xd5 0x5a 0xa9 0x78 0x30
                 0x6b 0x70 0x00 "
               }
               "w32-ix86"
               {
                 54 "0x61 0x84 0x00 0x00 0x00 0x30 0x02 0x04 0x00 0x00 0x00 0x00
                 0x02 0x04 0x00 0x00 0x00 0x01 0x02 0x04 0x00 0x00 0x00 0x01 0x61
                 0x84 0x00 0x00 0x00 0x18 0x02 0x04 0x00 0x00 0x00 0x10 0x04 0x10
                 0x02 0x00 0x25 0x16 0x09 0x35 0xc4 0xd8 0x00 0x00 0x00 0x00 0x00
                 0x00 0x00 0x00 "
               }
               "OBJECT_NIL" "tivoli:r)T!*'un"
               {
                 0
               }
               0 0 0 131077
             }

             {
               131077 0 "ishii" "w32-ix86" "C0F7NLVTS1CBVXH4CZDK00000593"
               0
               {
                 15 "0x15 0x5e 0x47 0xcd 0x31 0x92 0x1c 0xd5 0x5a 0xa9 0x78 0x30
                 0x6b 0x70 0x00 "
               }
               "NULL" 5 1956524575
               {
                 0 1 1
                 {
                   16 "0x02 0x00 0x25 0x16 0x09 0x35 0xc4 0xd8 0x00 0x00 0x00
                   0x00 0x00 0x00 0x00 0x00 "
                 }

               }

               {
                 0 0 0
                 {
                   0
                 }

               }

               {
                 0
               }

               {
                 1
                 [

                   {
                     0 1 1
                     {
                       16 "0x02 0x00 0x25 0x16 0x09 0x03 0xbb 0x84 0x00 0x00
                       0x00 0x00 0x00 0x00 0x00 0x00 "
                     }
```

6. Then the Endpoint Manager runs the `after_install_policy` by invoking the `after_install_policy` method.

```
loc-ic    67    M-H     1-53      287
     Time run:    [Wed 02-Dec 11:46:05]

     Object ID:   1189622596.1.517#TMF_LCF::EpMgr#
     Method:      after_install_policy
     Principal:   root@panda.itsc.austin.ibm.com (-2/-2)
     Path:        /var/spool/Tivoli/panda.db/methqdqJyc
     Input Data: (encoded):

          {
            8
            [
              "salmon" "1189622596.234.508+#TMF_Endpoint::Endpoint#"
              "w32-ix86" "1189622596.2.19#TMF_Gateway::Gateway#" "9.3.1.19
              3+9494" "1189622596" "234" "5"
            ]

          }

          {
            1
            [
              "LCF_LOGIN_STATUS=0"
            ]

          }

          {
            0
          }
```

7. Checking the return value of the `after_install_policy`. At this time, the `endpoint_login` method returns the result of the method invocation; then the Endpoint Manager runs the `update_endpoints` method, and the epmgr.bdb database is finally updated. The `update_endpoint` method is invoked each time the Endpoint logs into the Endpoint gateway, even if it is a normal login.

```
loc-ic    68 M-hdoq     2-47      420
      Time run:    [Wed 02-Dec 11:46:08]

      Object ID:   1189622596.1.517#TMF_LCF::EpMgr#
      Method:      update_endpoints
      Principal:   root@panda.itsc.austin.ibm.com (-2/-2)
      Path:        __epmgr_implid
      Input Data: (encoded):

             {
               1
               [

                 {
                   "salmon" "1189622596.234.508+#TMF_Endpoint::Endpoint#"

                   {
                     "FWQ7P94KRVTR0T02HHS300000577"
                     {
                       0
                     }
                     "OBJECT_NIL" "OBJECT_NIL" "OBJECT_NIL" 196608
                     {
                       15 "0x15 0x62 0x48 0x6f 0x7d 0x6f 0x1b 0xa8 0x78 0x43
                       0x79 0xf7 0x2c 0xc6 0x00 "
                     }
                     "w32-ix86"
                     {
                       54 "0x61 0x84 0x00 0x00 0x00 0x30 0x02 0x04 0x00 0x00
                       0x00 0x00 0x02 0x04 0x00 0x00 0x00 0x01 0x02 0x04 0x00
                       0x00 0x00 0x01 0x61 0x84 0x00 0x00 0x00 0x18 0x02 0x04
                       0x00 0x00 0x00 0x10 0x04 0x10 0x02 0x02 0x25 0x16 0x09
                       0x03 0x01 0xc1 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
                       "
                     }
                     "OBJECT_NIL" "tivoli:r)T!*'un"
                     {
                       0
                     }
                     4128 0 0 2
                   }

                 }

               ]

             }
```

8. The Endpoint receives the gateway assignment and logs into the appropriate Endpoint gateway.
9. Finally, to run the login_policy, the `login_policy` method is invoked on the Endpoint gateway, and the Endpoint initial login has been completed. The `login_policy` method is invoked each time the Endpoint logs into the Endpoint gateway even if it is a normal login.

```
loc-ic    50    M-H      2-3      252
      Time run:    [Wed 02-Dec 11:46:14]

      Object ID:   1189622596.2.19#TMF_Gateway::Gateway#
      Method:      login_policy
      Principal:   root@panda.itsc.austin.ibm.com (-2/-2)
      Path:        /var/spool/kodiak.db/methmvkXUa
      Input Data: (encoded):

             {
               8
               [
                 "salmon" "1189622596.234.508+#TMF_Endpoint::Endpoint#"
                 "w32-ix86" "1189622596.2.19#TMF_Gateway::Gateway#" "9.3.1.19
                 3+9494" "1189622596" "234" "5"
               ]

             }

             {
               0
             }

             {
               0
             }
```

As you can see, the `wtrace` command provides really detailed and useful information about the internals of the TMA operations. Most TMA actions can be checked using the `wtrace` command.

7.5 Endpoint Status File

The Endpoint provides a status file which contains status information for the Endpoint. By default, the status file (lcfd.st) should be located in the following directory.

UNIX	/opt/Tivoli/lcf/dat/1
Windows	C:\Program Files\Tivoli\lcf\dat\1

The following shows an example of the Endpoint status file.

```
state=INITIALIZING
timestamp=Feb:04:22:52:23
lcfdport=9494
```

The status field shows the current status of the Endpoint. This field can contain the following values.

- INITIALIZING
- NORMAL
- ISOLATED

- MIGRATE
- TMRREDIRECT
- IGNORED

Chapter 8. Overview of TMA Internals and Application Development

This chapter provides information about developing applications which support the TMA, such as:

- Design considerations
- TEIDL compiler
- Programming environment
- Runtime library for the TMA applications
- Dependencies
- Sample programs
- How to build applications

In this chapter the term LCF (Lightweight Client Framework) is used, as well as TMA, to follow the *Application Development for the Lightweight Client Framework* manual.

8.1 Application Design

As you have read in the previous chapters, the core features of LCF include:

- Three-tier structure - Endpoint Manager, Endpoint gateway and Endpoint
- Endpoints - small, dataless clients that have no client database and that do not have the full Tivoli Framework installed
- The scalability possible within the LCF environment - one gateway can manage thousands of Endpoints

An application must be designed and written based on these features. The application development environment for LCF is very similar to that of the full framework. However, there are some differences in the method invocation for the Endpoints. The following sections describe them.

8.1.1 Tivoli Object Methods

The next two sections briefly describe how methods work in both the traditional Tivoli Management Framework and in LCF.

8.1.1.1 Methods in the Full Framework Environment

In the full Framework Environment (TMR Servers and Managed Nodes), all methods are invoked in the same way. From any system within the TMR, a method can be invoked on any other system. The only thing you have to know for invocation is the object reference, which consists of the TMR Region Number, the Dispatcher Number and the Object Number.

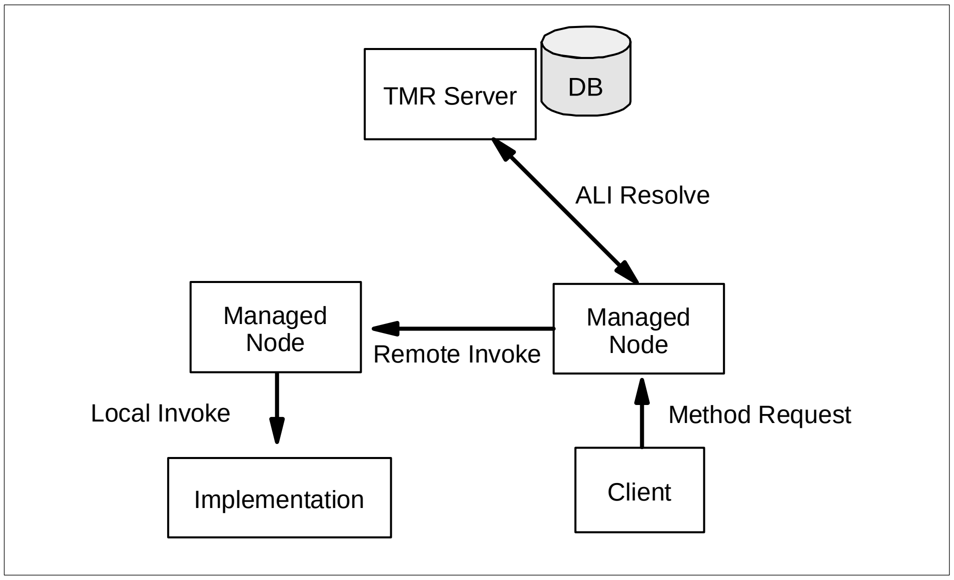


Figure 111. Remote Call in the Full Framework

The calling program passes the object reference to the Managed Node. The TMR Server is used to resolve it through the object dispatcher, locate the methods for that object, retrieve the method header, and invoke it.

The key point here is that any application on any Managed Node can invoke a method on any other Managed Node.

8.1.1.2 Methods in the TMA Environment

In the TMA environment, the method invocations are different from the full Framework environment. The differences are:

1. An application running on the Endpoint can only invoke methods on objects residing in its gateway. Restricting method requests from an Endpoint to its gateway allows the Endpoint to be simpler and smaller, since it does not need to resolve and locate remote objects. For this reason, an application that will require the Endpoints to manipulate remote objects needs the Endpoint gateway component. The Endpoint talks to this Endpoint gateway component, which has full access to the TMR to invoke the appropriate methods on the target object(s). This design enables the Endpoint to provide a higher degree of scalability compared with the previous versions of the Tivoli Management Framework.

2. In LCF, the method calls are different in the case of the upcalls (the methods invoked from the Endpoint) and the downcalls (the methods targeted for an object residing on an Endpoint).
3. The Endpoint methods use the tools and services of the Tivoli ADE, but they are implemented using a special application mini-runtime library (`libmrt`) specific to the LCF. The following are the new terms used in the LCF environment.

LCF Object	An object that runs on the Endpoint.
Endpoint Method	A method that runs on the Endpoint. The Endpoint method is invoked as the result of a downcall.
gateway Method	A method that runs on the Endpoint gateway. The gateway method is usually used as the result of an upcall made by the Endpoint associated with the Endpoint gateway.
Downcall	The method invocation from the gateway “down” to the Endpoint.
Upcall	The method invocation from an Endpoint "up" to the Endpoint gateway.

Note

An LCF object is an object running on an Endpoint. LCF object references have a special form:

R.D.P+ where:

- **R** is the TMR number.
- **D** is the number of the object dispatchers for the Endpoint.
- **P** is the number of the prototype object of the class for which the method is defined.
- **+** indicates the LCF object reference.

The object dispatcher numbers are uniquely identified for the Endpoint. The TMR server maintains the mapping of the Endpoint dispatcher numbers to the Endpoint gateway objects so it can route a request to the appropriate Endpoint gateway.

LCF objects are not created as instances in the object system and do not have a unique, per-object state maintained for them by the object system. Therefore, there is no need to create an instance of each object on the Endpoints as you do in the full Framework. Instead, they are born when the appropriate prototype object and the appropriate Endpoint dispatcher numbers are combined in this form. The Endpoint method runs on the Endpoint without calling the `create_instance` method.

To make use of LCF objects, the object reference is obtained from the registry or profile manager in the above form and invoked. The system locates the appropriate shared state (defined by the prototype object) and services the method request at the appropriate location.

Because there is no per-object data associated with an object by the system, applications have to assume responsibility for maintaining their own persistent store outside the context of the LCF-based services.

8.1.1.3 Endpoint Methods

An Endpoint method is a method that runs directly on the Endpoint. It is implemented using the special application mini-runtime library, `libmrt`, which provides a subset of the Tivoli operations.

When the Endpoint method is invoked on an Endpoint, the Endpoint spawner daemon uses the method executable stored in its cache. If the method is not in the cache or is not the most current version, the gateway will transfer the method’s executable to the Endpoint before invoking it. The Endpoint then adds the method to its cache for future use. The Endpoint methods differ from full Framework methods in the following ways:

- The TMA Endpoint is single-threaded. The Endpoint methods must be single-threaded, and only per-method methods are supported for the Endpoint methods. That is, each time a method is invoked on the Endpoint, a new instance of the method executable loads, executes, and terminates.
- There is no transaction support for Endpoints in the LCF environment.
- You must specify dependencies for Endpoint methods. The Endpoint method may be implemented across several files. Besides the executable containing the method, the method may require supporting files, such as shared libraries, message catalogs or other files. These supporting files are called dependencies and are defined as such in the method's definition. The Endpoint gateway will ensure that all dependencies for the method have been downloaded to the Endpoint before invoking the method.

In the full Framework, only the method body, the binary program or script that contains the method entry point is stored in the method header. Any supporting files that the method requires are assumed to be present. This is true because all binaries, library, message catalogs and so on are installed on every Managed Node and TMR server.

LCF Endpoints do not have any methods or supporting files present on them when the Endpoints are initially installed. The method bodies are identified in the standard method header and are downloaded when they are needed. Because the dependencies (the supporting files) must also be downloaded when they are missing, the dependencies must be called out in the Endpoint method so they can be present when they are needed.

8.1.1.4 Gateway Methods

A gateway method runs on the Endpoint's assigned gateway. It runs as the result of an upcall request from the Endpoint or as the result of the request from another Managed Node. Since the Endpoint cannot communicate directly with other nodes, it must initiate all actions by invoking the upcall on the gateway method. The gateway method then takes advantage of the full Tivoli Management Framework to resolve and locate the appropriate target object(s) and invoke the proper methods.

8.1.2 Downcalls and Upcalls

This section describes the activity involved with upcalls and downcalls.

8.1.2.1 The Sequence of a Downcall

The process whose method request originates on the Managed Node and is executed on the Endpoint is called a downcall. The Endpoint receives a program name to execute, any arguments and runs the program. After the Endpoint method completes, the Endpoint returns the result back through the gateway. If the requested Endpoint method does not already exist on the Endpoint or if the method on the Endpoint is out of date, the method executable is downloaded to the Endpoint. If the method has dependencies, they are also downloaded.

The following brief sequence summarizes the downcall from the Managed Node to the Endpoint:

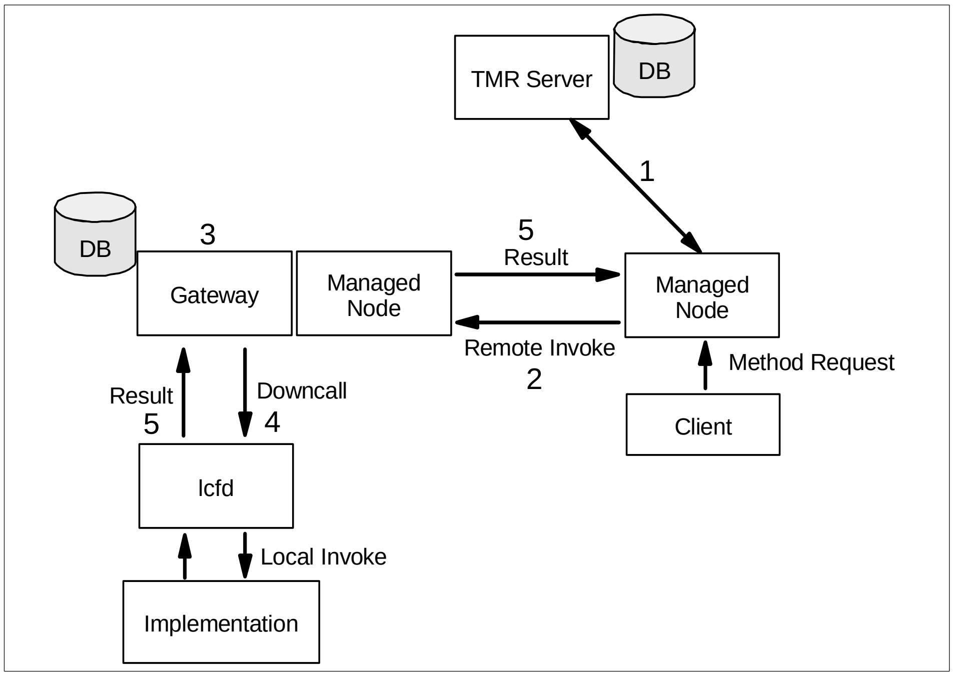


Figure 112. The Downcall Processes

1. The TMR server maps the dispatcher in the object to the gateway object ID. The gateway object ID is used to determine which Managed Node the gateway is running on. In this case, the plus sign (+) in the object reference designates that the object runs on the Endpoint.
2. The TMR object dispatcher sends the request to the gateway servicing the Endpoint.

3. The gateway resolves the method on the corresponding behavior object and performs authorization of the invocation by invoking the Tivoli principal.
4. The method runs on the Endpoint with the downcall. The parameters are sent; then the method runs, and the MDist data is transmitted from the gateway if it is needed.
5. The results are passed back to the gateway. The gateway then returns the results to the caller.

8.1.2.2 The Sequence of Upcall

Not all Endpoint applications need to make upcalls. However, if the Endpoint application attempts to invoke a method located in a Managed Node, the Endpoint makes the upcall by calling an IDL compiler-generated stub. Instead of passing an object reference to the stub function, the caller passes the class name.

Unlike method invocation in the full Framework, the upcall always runs the gateway method on the host machine of the gateway. The application that supports upcalls must provide both the Endpoint code and the gateway upcall implementation.

The gateway attempts to resolve the method and perform necessary authorization without making the call to the object dispatcher on the TMR server. The method header can be retrieved from the method header cache, as it is for downcalls. The object groups for an LCF object are determined on a per-Endpoint basis; that is, the Endpoint is placed in a security group, and all objects on that Endpoint are members of the same security group.

The Endpoint gateway tells the Managed Node object dispatcher to start the gateway method; the object dispatcher starts the daemon for the method, if not already started, and the daemon then proceeds. The results are returned through the gateway daemon back to the Endpoint.

The following figure illustrates the flow-of-control when the Endpoint performs the upcall by invoking the gateway method.

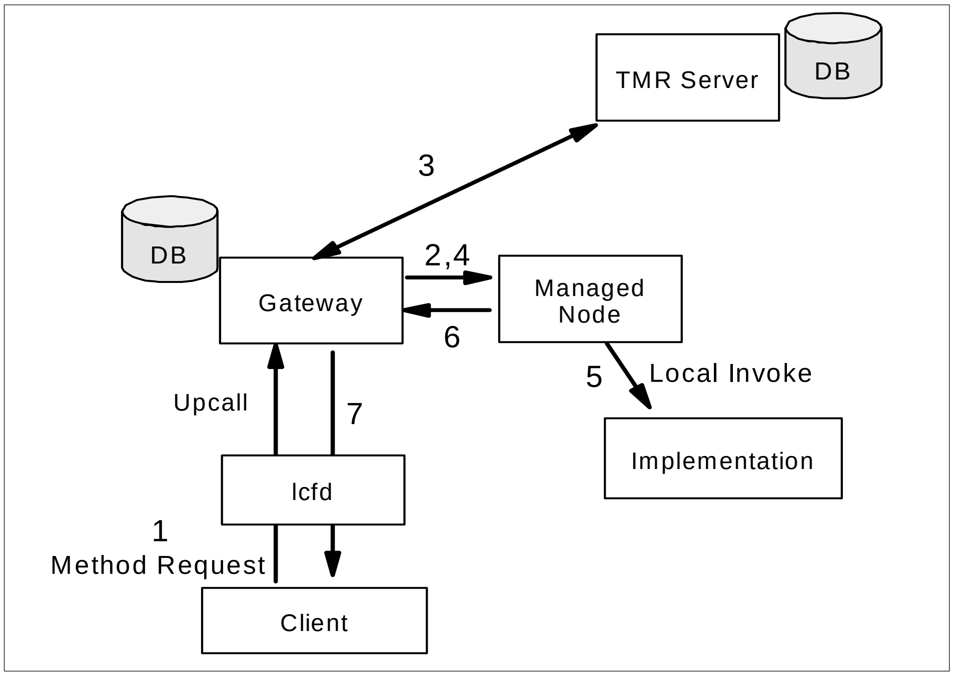


Figure 113. The Upcall Processes

1. The Endpoint sends the request to the gateway.
2. If the gateway can resolve the method header, it tells the object dispatcher on the Managed Node to invoke the method.
3. If it cannot resolve the method header, it goes to the TMR Server to get the header.
4. When the header is returned to the gateway, the gateway tells the managed node to invoke the method.
5. The method is invoked on the Managed Node.
6. Results are passed back to the gateway.
7. The gateway passes the results back to the Endpoint.

Each time your application makes an upcall, it contacts lcfd, which seals the upcall data and sends it to the gateway. The data is sealed so that it cannot be modified on the wire without detection, but it is visible on the wire. Only keys in the data are encrypted for privacy.

8.1.2.3 Making Upcalls: Contacting lcfd Daemon

The Endpoint method implementations can make upcalls without contacting the LCF daemon (`lcfd`). However, client programs need to contact `lcfd` daemon to make upcalls. To contact `lcfd` daemon, the client program must know the communication settings that the `lcfd` daemon uses and be able to determine the port that the `lcfd` daemon uses. To run upcalls, you need:

- The $LCF_DATDIR directory

  or

- The appropriate lcf_env.sh source file. The lcf_env.sh file resides in the uniquely numbered subdirectory in the known location on each interp.

8.1.2.4 Upcall Authorization

In the full Framework environment, Tivoli uses the operating system user authentication and then maps operating system users to Tivoli principals. However, Endpoints may vary in their native functionality. They may not have a multi-user operating system; they may not be secure, and so on.

To make authentication easier for the application, Endpoints have the inherent capability to invoke methods that have ANY_ACL in their access control list (ACL). This is useful for applications that perform asynchronous upcalls; they can provide benign information to the application. (Examples of this type of application are Tivoli Distributed Monitoring and Tivoli Inventory.) The security of applications that choose to take advantage of this feature must be carefully designed in the following ways:

- Methods called by Endpoints must have ANY_ACL in their ACL list.
- Methods called by Endpoints must have high enough roles to invoke methods.

The application designers should ensure that cascaded object calls performed by gateway method implementations do not have security risks, since they can be indirectly invoked by anyone.

8.1.3 Scalability Considerations for TMA Applications

This section discusses considerations to keep in mind when you design applications for the LCF environment and describes the components of an LCF application. When designing your LCF application, pay attention to the following points.

- The scalability of applications in the LCF. If an operation occurs on one Endpoint, it probably could occur on hundreds or thousands of Endpoints at the same time. Design your application to use the scalability features that are part of the LCF. This includes, for upcalls, using an upcall collector daemon to collect and forward data. For downcalls, use the MDist repeater capability the gateway provides. Please refer to the *Application Development for the Lightweight Client Framework* for more conceptual information about the upcall collector.

- There is no database or other persistent store on the Endpoints.
- You must consider the functionality needed on both the client and server sides of your application.

The fact that Endpoints are dataless affects application design. Except for the LCF daemon (`lcfd`) itself, there is no framework application database or other application-specific state kept on the Endpoint.

The following are the scalability considerations you should keep in mind when design your LCF application.

- Minimize the Endpoint application footprint.
- No persistent store on Endpoints.
- Data flows to and from the Endpoints.
- Use the three-tiered design.

8.1.3.1 Minimize Endpoint Application Footprint

Be sure your application's Endpoint code is no larger than it needs to be. Make full use of `libmrt`, adding only code that is necessary. Be aware that your application's size affects the following.

- Space required on the Endpoint gateway
- Space and download time for the Endpoint

Method implementations and dependencies are downloaded to Endpoints on demand. Endpoint binaries and shared libraries are downloaded from the gateway's implementation repository. The larger they are, the more space is required on the gateway's implementation repository to store this Endpoint code. Remember that Endpoint methods and dependencies for each interpreter type are stored on each gateway.

The larger the Endpoint code, the longer it takes to download methods and dependencies to Endpoints. After they are downloaded, each binary and shared library competes for Endpoint disk cache space. This means that, if the cache is full, a binary or shared library may be removed and have to be downloaded the next time it is needed.

8.1.3.2 Storing Data

There should be no state on an Endpoint or gateway unless it is temporary, transient, or uses the cache. Any Endpoint-specific data that needs to be accessed by entities other than that Endpoint should not be stored on the Endpoint. The most common place to store information is in a RIM repository, although the object database may also be used.

Remember that Endpoints are targets of distributions, sources of events, and so on. Communicating with Endpoints to find information that has been stored has a cost: Communication over wide, slow, disparate networks is expensive.

It is important to minimize the network crossings of data. Data stored across these networks is difficult to access; if there are external references to it or copies of it, it is even harder to keep consistent.

8.1.3.3 Data Flow to and from Endpoints

Data sent to multiple Endpoints should use MDist to take advantage of the MDist repeater fan-out and network bandwidth management capabilities that are part of the gateway. Because the framework provides no way to fan-in data, applications that send data “up” should do so in a scalable manner. Large amounts of data should be sent by only a few Endpoints at a time. For example, if your application is polling data, do not activate a poll on more than a few Endpoints at a time.

8.2 Introduction to Tivoli ADE Extended IDL

In the LCF environment, CORBA IDL is used to define object interfaces. It is the same case with development in the full Framework. The TEIDL compiler that is used for the full Framework is used in the LCF environment. We will discuss the TEIDL compiler in this section.

According to the CORBA specification, the purpose of the IDL is to provide a standard for defining interfaces that client objects call and object implementations provide. It does not, however, provide a standard for implementation or installation.

Thus, when you define your interface in IDL, be aware that not only method implementation, but also a program that generates and installs a class hierarchy must also be generated.

Tivoli ADE therefore provides the following extensions to the OMG IDL.

• Implementation

- Implementation inheritance
- Installation and initialization
- Hierarchical exception classes

These extensions are provided in a parallel, but separate, language: Tivoli Extended IDL (TEIDL). The grammar for TEIDL follows the same lexical and syntactic conventions as those specified by CORBA for IDL.

8.2.1 TEIDL Compiler Input

When you develop your application using TEIDL, you should create the following files that define, implement, and install your interface:

- The interface definition file, which has a name containing an extension of .idl.
- The implementation specification file, which has a name containing an extension of .imp.
- The program specification file, which has a name containing an extension of .prog.
- The installation specification file, which has a name containing an extension of .ist.

For more information about TEIDL input files, please refer to 8.7.4.1, “The IDL Files” on page 324.

8.2.2 TEIDL Compiler Output

When you have defined an IDL interface, you must invoke the TEIDL compiler to produce the necessary source code for your interface. The TEIDL compiler produces the following output (excluding the N/A item):

Table 48. TEIDL Output

Name	File Name	Type	Linkage			
			CORBA		TME	
			Client	Svr	Client	Svr
Public Header	`<pfx1>.h`	CORBA	X	X	X	X
	`t_<pfx1>.h`	TME			X	X
Defines Header	`<pfx1>_defs.h`	COMMON	X	X	X	X

Name	File Name	Type	Linkage			
			CORBA		TME	
			Clie nt	Svr	Clie nt	Svr
Auxiliary Header	`<pfx1>_aux.h`	COMMO N	X	X	X	X
Auxiliary Source	`<pfx1>_aux.c`	COMMO N	X	X	X	X
Stub Source	`<pfx1>_stub.c`	CORBA	X			
TME Stub Wrapper	`t_<pfx1>_stub.c`	TME			X	
N/A		Appl. Main	X		X	
Main Program	`<pfx3>.c`	COMMO N		X		X
Private Header	`<pfx1>_imp.h`	CORBA		X		X
	`t_<pfx1>_imp.h`	TME				X
Private Source	`<pfx1>_imp.c`	CORBA		X		X
	`t_<pfx1>_imp.c`	TME				X
Skeleton Source	`<pfx2>_skel.c`	CORBA		X		X
TME Skeleton Liner	`t_<pfx2>_skel.c`	TME				X
Method Template	`<pfx2>_meth.c`	CORBA		X		
	`t_<pfx2>_meth.c`	TME				X
Install Script	`<pfx1>.cfg`	COMMO N				
Install tar File	`<pfx1>_ist.tar`	COMMO N				
IR tar File	`<pfx1>_ir.tar`	COMMO N				

- `pfx1`: prefix of input file name
- `pfx2`: prefix of input file name or program name
- `pfx3`: program name

Note

This table contains the CORBA and TME Client/Server linkage requirement information. This information is for the full Framework side of client/server. For the Endpoint side, please refer to 8.8.1, “Sequence of Steps for Building a TMA Application” on page 343.

The TEIDL compiler produces three sets of files as follows.

- TME - Used for applications that call Tivoli stubs
- CORBA - Used for applications that call CORBA stubs
- Common - Used to implement and install any applications regardless of which stubs they call

When you design your applications, you have the option of providing clients that use Tivoli stubs and skeletons or CORBA stubs and skeletons. Subsequent sections describe the difference between these files.

8.2.2.1 Tivoli (TME 10) Output

Tivoli ADE provides several services that make it easier to write distributed applications. You should use the TME 10 files generated by the TEIDL compiler if you plan to use either of these services.

- TME 10 Nested Transactions
- TME 10 Exceptions

Please refer to the *Framework Service Manual* nested transactions and exceptions.

The TEIDL compiler generates the following TME 10 files.

TME 10 private header	Contains private APIs for TME 10 applications
TME 10 method template	Contains templates for methods that use TME 10 nested transactions or TME 10 exceptions
TME 10 public header	Contains TME 10 APIs for client use
TME 10 private source	Contains definitions of accessor and mutator functions implemented in methods that use

	TME 10 nested transactions and TME 10 exceptions
TME 10 skeleton liner	Contains TME 10 skeleton liners that interface with the TME 10 CORBA skeleton
TME 10 stub wrapper	Contains definitions of TME 10 stub wrappers, which are declared in the TME 10 public header

The TME 10 stub wrappers and skeleton liners contain code wrapped around CORBA stubs and CORBA skeletons. They are thus CORBA compatible and can call, and be called from, CORBA-compatible stubs.

You should not modify the files listed in this section, except for the TME 10 method template(s). The method template files contain only the required preprocessor statements and empty method bodies.

Use these template files to implement the methods required for your application. You need to be aware of the other TME 10 files for the purpose of compiling and linking your application.

8.2.2.2 CORBA Output

If you do not want to use TME 10 nested transactions or TME 10 exceptions, you should use the following CORBA files produced by the TEIDL compiler

Private header	Contains private APIs for CORBA applications
Method template	Contains templates for methods that do not use TME 10 nested transactions or TME 10 exceptions
Public header	Contains CORBA APIs for client use
Private source	Contains definitions of accessor and mutator functions implemented in methods that do not use TME 10 nested transactions and TME 10 exceptions
Skeleton source	Contains the skeletons for object implementations that do not use TME 10 nested transactions or TME 10 exceptions
Stub source	Contains definitions of stub functions declared in the CORBA public header

You should not modify the files listed in this section, except for the method template(s). The method template files contain only the required preprocessor statements; you use them to implement the methods required for your application. You should be aware of the other files for the purpose of compiling and linking your application.

8.2.2.3 Common Output

The following TEIDL compiler-generated files are used by all applications, regardless of whether they use TME 10 nested transactions or TME 10 exceptions.

Configuration script	Contains the commands necessary to install your application
Installation tar file	Contains information for the installation script
Auxiliary header	Contains the following:

- Function declarations for compiler-generated marshal routines
- Marshal tables for each IDL operation and attribute

Auxiliary source	Contains the following:

- TypeCode constant definitions
- Type repository information
- Definitions of compiler-generated marshal routines
- Definitions of copy/equal functions for interface types

Defines header	Contains `#define` for operation and attribute names used by the BOA to locate the correct entry point in an implementation
Main program	The main program for the object implementation

8.2.3 The Stub and the Skeleton

In the CORBA environment, the client program calls the stub routine, and the server method implementation is called from the skeleton routine. This communication is managed by the ORB runtime.The following figure illustrates the flow of the normal remote method call.

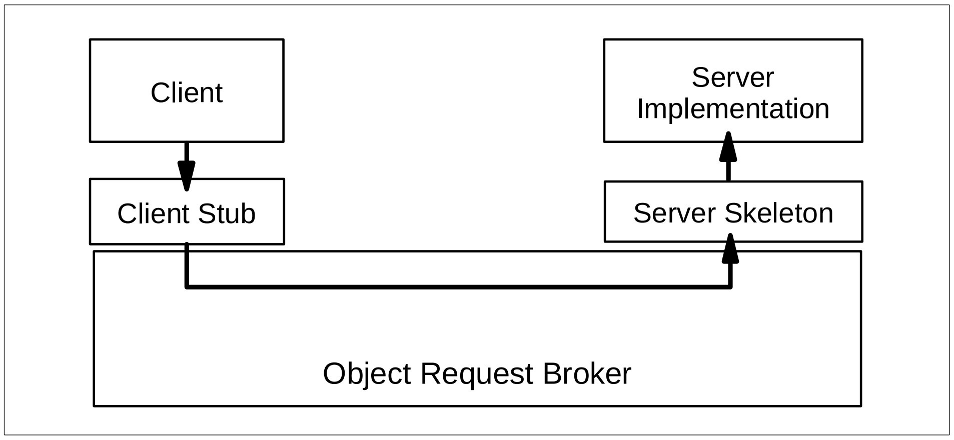


Figure 114. CORBA Stub and Skeleton

Note

In the CORBA environment, the CORBA method requester is called the client, and the method implementation that provides service is called the server. In this case, the client does not mean Endpoint. In the same way, the server may become a client at the next cascaded call.

The purpose of the CORBA client stub is to manage the transfer of data from the client program to the ORB and to pass returned data back to the client. In this sequence, it performs two hidden tasks:

- Marshals the parameters passed to it as part of request
- Unmarshals the return data from the object implementation

If you are using TME 10 transactions and exceptions, the stub must also manage these items. The compiler therefore produces a TME 10 wrapper for each CORBA-compliant stub, which facilitates the management of TME 10 transactions and exceptions.

The purpose of the CORBA server skeleton is to manage the transfer of data from the ORB to the object implementation and to manage the return of the data to the ORB. In this sequence, it performs two additional tasks:

- Unmarshals the request from the client
- Marshals the return data from the object implementation

If you are using TME 10 transactions and exceptions, the skeleton must also manage these items. The compiler therefore produces the TME 10 liner for each CORBA-compliant skeleton, which facilitates the management of TME 10 transactions and exceptions.

8.2.4 Method Templates

The Tivoli ADE extended compiler produces two C source file templates for each method specified in your implementation files: a TME 10 template and a CORBA template. You should implement your method using only one of these two files.

Before you implement your method, choose the appropriate method template. If your method uses TME 10 exceptions or TME 10 transactions, implement it using the TME 10 method template whose name has the following form: t_<pfx2>_meth.c, where <pfx2> is the name of the implementation file.

If your method does not fit into one of the presented categories, implement it using the standard CORBA method template whose name has the following form: `imp-file-name_meth.c`, where `imp-file-name` is the name of the implementation file.

The following code fragment depicts the typical TME 10 method template:

```
/*
 ****************************************************************
 *
 *   File Name: t_Upcall_main_meth.c
 *   Tivoli EIDL Compiler (Version 2.0, Date 09/19/97)
 *   generated ANSI C Tivoli method implementation File.
 *
 *   Edit this file to fill in method implementations.
 ****************************************************************
 */
#include <tivoli/t_Upcall.h>
#include <tivoli/t_Upcall_imp.h>
#include <tivoli/ExException.h>

void t_imp_LCF_iUpcall_method(
    LCF_Upcall _LCF_Upcall,
    Environment *_ev,
    transaction _transaction,
    char * input,
    char ** output) {
}
```

The original IDL file `Upcall.idl` and the implementation file `Upcall.imp` are shown in 8.7.4, “The Upsamp Files” on page 324.

The template provides the necessary preprocessor directives and the operation with a signature which contains the parameter that specifies the transaction type. The compiler produces the template contents from the information in your IDL and TEIDL files.

The method template contains the directive to include `ExException`, which enables the method to catch and throw exceptions derived from this class. This is different from the CORBA method template, which does not contain the reference to any TME 10 exception class.

After you run the TEIDL compiler, you can open the file and edit it using a standard editor such as `vi` or `emacs`. The TEIDL compiler also produces the complementary header file, but it is complete. Therefore, you don’t need to modify it.

8.2.5 Configuration Script

When you run the TEIDL compiler, it produces the configuration script which contains the commands for installing the classes defined in your TEIDL files. After running the compiler, you should run this script before attempting to instantiate any classes or initiate requests of any objects.

When you run the script, it will add the appropriate class references to the library in the Tivoli Management Region (TMR). By these references, when the service of the class is requested, the ORB can correctly identify the class and pass the request to it. See the *Application Services Manual* for information about TMRs.

After running the script, your class is installed and initialized. You can then create class instances and initiate requests of those instances as required by your program.

All Tivoli classes in each TMR are registered in the name registry for the TMR by the configuration script. The Tivoli Name Registry (TNR) is the primary means for managing object references, user-friendly names for classes, and instances of those classes. See the *Application Services Manual* for more information about the name registry.

8.2.6 Building a Client-Server Program

Table 48 on page 292 lists the files required to build your client-server program. You must compile and link indicated sources together to build client and server application. Please refer to the 8.8, “Building the Sample Application” on page 342 for more information about the build operation.

8.3 Tools for Endpoint Applications

The tools needed by applications are included in the Endpoint gateway repository, $BINDIR/../lcf_bundle/bin/$INTERP/tools. The applications (for example, the task library) that have need to access the tools such as the Bourne Shell (sh.exe) and Perl (perl.exe) on the Endpoint must establish the dependency on the gateway to cause the `lcfd` daemon to download the tool(s) on demand.

To establish this dependency, first create a dependency set (with the `wdepset` command) and then associate it with a method (with the `wchdep` command). For information about how to get tools to the Endpoint, see “Using Dependencies to Deploy Tools to Endpoints” on page 318.

8.3.1 LCF Environment for Methods and Tasks

This section discusses the available environment when the `lcfd` daemon spawns the method or task.

Certain environment variables are exported for the methods and tasks that the `lcfd` daemon spawns. When the `lcfd` daemon spawns a method or task on the Endpoint, these variables determine the environment in which the spawned process runs. The following variables, which are set when the `lcfd` daemon starts, are available for use by methods and tasks.

LCF_BINDIR	Directory for Endpoint binaries.
LCF_CACHEDIR	Cache directory for the Endpoint. This directory contains methods and dependencies that downloaded by the `lcfd` daemon for the Endpoint application.
LCF_DATDIR	Directory for Endpoint configuration information (last.cfg) and log files.
LCF_LIBDIR	Directory for Endpoint libraries (`libmrt`, `libcpl`, and so on).
LCF_TMPDIR	Temporary directory. This directory can be used by methods to store data.

These variables cannot be changed dynamically. All values are in absolute paths and are obtained from these sources in the following order:

1. ***The default se*t:** The programmatic defaults are set when the `lcfd` daemon starts. The applications cannot change these defaults.
2. ***The last.cfg fil*e:** You can override the default set by the `lcfd` for $LCF_CACHEDIR by changing the value in this file; the change is persistent across the system boots.

3. ***The command line:*** You can override the default set by using the `lcfd` or `lcfd.sh` commands.

The default paths are determined by the install configuration of the product. The base path which is set in the installation, such as the /Tivoli/lcf, the install program builds the other paths on this basis. For example, on Windows NT, when you set the base path to C:\Tivoli\lcf, the defaults would be:

```
LCF_BINDIR=C:\Tivoli\lcf\bin\w32-ix86\mrt
LCF_LIBDIR=C:\Tivoli\lcf\lib\w23-ix86
LCF_DATDIR=C:\Tivoli\lcf\dat\1
LCF_CACHEDIR=C:\Tivoli\lcf\dat\1\cache
LCF_TEMPDIR=C:\temp
```

8.3.2 LCF Environment for CLIs

You can setup the environment variables on the Endpoint that enables you to run CLIs on Endpoints. You set these environment variables by implementing them in the setup script. These variables can then be used by an application when it runs on an Endpoint. These variables can also be passed by an application or by the CLI to the `lcfd`.

8.3.2.1 Setup Scripts for CLIs on Endpoints

The environment setup file for Endpoints, lcf_env.*, sets up the environment on the Endpoint so that applications running on the Endpoint can use CLIs, do upcalls, and so on. These environment variables define directory locations and other environment information for the LCF Endpoints and are set by the environment setup scripts, analogous to the setup scripts (setup_env.*) for the Tivoli management applications. The users don't need to source the environment setup script to run the `lcfd`, but applications need them.

8.3.2.2 Environment Variables Set in the Scripts

The scripts set up environment variables for paths on the Endpoints, such as the path to the root directory and the directory for libraries. There are two user-configurable paths on the Endpoint: the LCF root directory (the top directory of the LCF installation) and the path to the cache directory. When the `lcfd` is installed, all other paths are set relative to these directory locations. The environment variables in the setup scripts contain the directories that are set relative to the LCF root directory and the cache directory. The following describe the variables set by the environment setup scripts.

LCF_BINDIR	The directory for Endpoint binaries. Contains `lcfd`. Set to $LCFROOT/bin/$INTERP/mrt by the setup script.

D R A F T

LCF_CATDIR	The directory for message catalogs on the Endpoint. Set to $LCFROOT/msg_cat by the script.
LCF_DATDIR	The directory for the Endpoint configuration file (`last.cfg`) and the log file.
LCF_LIBDIR	The directory for Endpoint libraries (`libmrt`, `libcpl`, and so on). Set to $LCFROOT/lib/$INTERP by the setup script on platforms (such as UNIX) that use separate search path environment variables for executables and shared libraries. Set to $LCF_BINDIR for platforms (such as PCs) that use the same search path environment variable for both executables and shared libraries.
LCF_TOOLSDIR	The directory, by convention, where the out-of-cache tools are placed when downloaded to the Endpoint. LCF_TOOLSDIR is set to $LCFROOT/bin/$INTERP/tools.
LCFROOT	The root directory for the Endpoint: the directory in which the user installed the LCF.
INTERP	The interpreter type for the Endpoint. The available values are shown below.

Table 49. Interpreter Types

Operating System	Interpreter Type
AIX 3.2.5	aix3-r2
AIX 4.1.2 through 4.1.5, 4.2, and 4.3	aix4-r1
HP/UX 9.00 through 9.07	hpux9
HP/UX 10.01, 10.10, and 10.20	hpux10
HP/UX 11	hpux10
NetWare 3	nw3
NetWare 4	nw4
OS/2	os2-ix86
OS/390	os390
OS/400 V3R2, V3R7, V4R1, and V4R2	os400
Solaris 2.3, 2.4, 2.5, 2.5.1, and 2.6	solaris2
SunOS 4.1.2, 4.1.3, and 4.1.4	sunos4
Windows NT 3.5.1 and 4.0	w32-ix86

Operating System	Interpreter Type
Windows 3.x	win3x
Windows 95	win95

NLSPATH	The directory used by applications to determine the language to use. $NLSPATH is appended to $LCF_CATDIR/$L/$N.cat;$NLSPATH by the script.
PATH	The path variable. `PATH` is added to $LCF_BINDIR.
TISDIR	Sets to $LCF_DATDIR by the script. At normal login, lcfd checks to see if it has its codeset file; if not, the gateway sends it. The codeset file is written to $LCF_DATDIR/xxxx, where xxxx is the name of the codeset the `lcfd` requested. For example, the codeset for a Solaris 2.x box running in English is ISO88591.

Some platforms use the same search path environment variables for both executables and shared libraries. Other platforms, such as UNIX, use separate search path environment variables for the executables and the shared libraries as follows.

`LD_LIBRARY_PATH`	The variable for the shared libraries on Solaris 2 and SunOS 4. `LD_LIBRARY_PATH` is added to $LCF_LIBDIR.
`LIBPATH`	The variable for the shared libraries on AIX 3.2 and AIX 4.1. `LIBPATH` is added to $LCF_LIBDIR.
`SHLIB_PATH`	The variable for the shared libraries on HPUX 9 and HPUX 10. `SHLIB_PATH` is added to $LCF_LIBDIR.

8.3.3 Debugging Endpoint Method

There are two ways to debug Endpoint methods in the LCF environment:

- Use the ADE debugging tools from the full Framework, such as the `wdebug` command. With this tool, you can debug one method at a time.
- Use the `-D debug_flags=1` option to start the `lcfd`. With this option, you can stop each Endpoint method when it starts and attach the debugger. In this way, you can debug all methods on the Endpoint. For example, to debug you can use the command:

```
lcfd -D debug_flags=1
```

When each method starts, the `lcfd` prints to the console the method name, and the process it starts. It then suspends the methods so you can attach the debugger.

8.4 Application Runtime Library

The LCF application mini-runtime library (`libmrt`) contains functions you use to implement Endpoint methods. This library provides the smallest subset of library functions needed to implement the LCF method executable for the Endpoint. Using this special application runtime library helps to keep the Endpoint portion of the application as small and simple as possible. The `libmrt` contains subsets of the following functions provided by the full Framework:

- Memory management
- Distributed exceptions
- Sequence manipulations
- File system input/output
- Logging functions
- ADR marshalling functions

The following sections summarize the functions available in the `libmrt` library.

8.4.1 Memory Management

The library provides the subset of memory management functions that works with either local or global memory:

- Use local memory for the temporary allocations that are automatically freed after use.
- Use global memory for the all allocations that must be persistent.

The table below summarizes the memory management functions available for Endpoint methods.

Table 50. Memory Management Functions

Routine Name	Description
mg_malloc	Allocates a block of uninitialized global heap memory
mg_free	Frees global memory allocated by mg_malloc mg_calloc, mg_strdup, or mg_realloc
mg_calloc	Allocates a block of global memory, initialized to zero

Routine Name	Description
mg_realloc	Reallocates global memory; changes the size of an allocated block of memory
mg_strdup	Copies a string into a new block of memory allocated with mg_malloc
mg_cleanup	Frees all globally allocated memory (mg_*alloc) that has not been deallocated
ml_create	Creates a new local memory heap
ml_malloc	Allocates uninitialized local memory
ml_free	Frees local memory allocated with the ml_*alloc functions or with ml_strdup
ml_calloc	Allocates local memory, initialized to zero
ml_realloc	Reallocates local memory
ml_to_mg	Moves memory from local to global
ml_strdup	Duplicates string using ml_malloc
ml_destroy	Frees all memory in a local memory heap
ml_ex_malloc	Allocates uninitialized local memory, in a Try() frame
ml_ex_calloc	Allocates local memory, initialized to zero, in a Try() frame
ml_ex_realloc	Reallocates local memory, in a Try() frame
ml_ex_strdup	Duplicates string using ml_ex_malloc, in a Try() frame

8.4.2 Distributed Exceptions

Exceptions are used to report the fatal error. You can use the standard Try/Catch frame macros or the variable argument exception functions to handle exceptions. The table below shows the Try/Catch frame macros that you can use.

Table 51. Try/Catch Frame Macros

Macro	Description
Try	Starts a new Try/Catch frame

Macro	Description
Catch	Catches an exception of the given type
CatchAll	Catches exceptions of any type
EndTry	Ends a Try/Catch frame
Throw	Throws an exception
ReThrow	Rethrows a caught exception
ev_to_exception	Converts an environment to an exception
exception_to_ev	Converts an exception to an environment

In some cases, you might find it is easier to use the variable argument (`printf`-style) exception functions rather than the Try/Catch macros. The table below shows the variable argument exception functions that you can use.

Table 52. Variable Argument Exceptions

Routine	Description
vaThrow	Throws an error message
vaThrowDerived	Throws a type of error message
vaMakeException	Returns a pointer to the exception
vaAddMsg	Appends a new message to an X/Open message
ThrowExErrorMsg	Throws a message as an exception

8.4.3 Sequence Manipulations

The data types supported by the IDL do not include variable length arrays. Instead, you must use the data type called sequence. The sequence consists of a pointer to an array of the given data type and the count of the number of elements in the array.

The LCF environment supports a limited subset of functions available to manipulate sequences, and its usage is slightly different from the one used in the full Framework. In the full Framework, the sequence APIs are all implemented as function calls in libraries. You must set all references of user-defined sequences from the native data type to the sequence_t type. In the LCF, the sequence APIs are lightweight macros defined in seq.h. In the LCF, it is no longer necessary to cast from the user-defined type to the sequence_t type. All the LCF sequence macros allow you to use the sequence without setting values to and from sequence_t. The table below shows the supported sequence macros, where <type> is a typed sequence_t.

Table 53. Sequence Manipulations

Sequence Macros	Description
Seq_new(size_t size)	Creates the sequence of elements of the size specified in memory
Seq_zero(<type> *seq)	Clears the sequence.
Seq_len(<type> *seq)	Returns the number of elements in the sequence specified
Seq_get(<type> *seq, int index)	Returns the pointer to the data item in the sequence for the index specified.
Seq_add(<type> *seq, <type> item)	Adds the data item to the end of the sequence
Seq_remove(<type> *seq, int index)	Removes the data item from the sequence
Seq_free_buffer(<type> *seq)	Frees the memory allocated for the buffer portion of the sequence specified

8.4.4 File System Input/Output

The LCF supports a set of functions for the file system input and output as the table below summarizes.

Table 54. File System Input and Output

Routine	Description
open_ex	Opens a file. Throws an exception on error
read_ex	Reads a file. Throws an exception on error
write_ex	Writes a file. Throws an exception on error
close_ex	Closes a file. Throws an exception on error

Routine	Description
makedir_ex	Makes a directory. Throws an exception on error
make_path	Checks for and builds every component of the path
does_file_exist	Returns true or false
get_file_length_ex	Returns the number of bytes in a file
get_open_file_length_ex	Returns (in bytes) the size of a file opened with open_ex
lseek_ex	Moves around in a file opened with open_ex
rename_file_ex	Renames a file. Throws an exception on error
copy_file_ex	Copies source path to destination path and returns the number of bytes copied. Throws an exception on error
ep_stream_read	Reads an MDist stream

8.4.5 Logging Functions

The logging utility provides functions that enable you to create multiple logs to produce print-style messages to the console and to a file as the table below summarizes.

Table 55. Logging Functions

Routine	Description
LogInit	Creates a new log file. It also backs up the old log file and allocates resources needed by the log module.
LogDeinit	Deallocates resources set by a call to LogInit.
LogMsg	Uses the Tivoli National Language Support (NLS) to format an internationalized message and then outputs it to the console and log file
LogSetDefault	Maintains a static (private) pointer to the default log structure

Routine	Description
LogGetDefault	Returns a pointer to the default log structure
LogSetThreshold	Sets the output threshold of the display level of the requested log
LogGetThreshold	Returns the value of the display_threshold for that log
LogSetOutputStdout	Sets the Boolean to output messages to stdout
LogGetOutputStdout	Returns the values of the output_stdout for that log
LogSetAppName	Sets the identifier to be used in logging messages
LogGetAppName	Returns the value of the application name for the requested log
LogQ	Is a wrapped function around LogMsg; it implements a circular queue in memory of the last n messages, to be included in exceptions
LogQueueAlloc	Allocates the size of the buffer LogQ messages
LogQueueDealloc	Shuts down and deallocates memory used for LogQ buffer
LogQueueGetSize	Queries the size of the buffer for LogQ messages
LogQGetBuffer	Returns a character array containing the circular queue of LogQ messages
LogData	Formats and logs binary data

8.4.6 ADR Marshalling Functions

The abstract data representation (ADR) functions are identical with the full Framework version, except for the fact that the LCF Endpoint does not have the interface repository. It does not have the dynamic data type lookup; so all data types must be present at compile time. If you need IDL types on an Endpoint, they must be compiled into the application’s module. The Interface Repository (IR) is a service that provides persistent objects that represent the IDL information in a form available at run-time. The IR may be used by the ORB to perform requests. Moreover, using the information in the IR, the program may encounter an object whose interface was not known when the program was compiled; yet, it is able to determine what operations are valid on the object and make an invocation on it. In addition to its role in the functioning of the ORB, the IR is suitable for storing the additional information associated with interfaces to the ORB objects. The following data types are built in to the `libmrt`:

- any
- boolean
- char
- double
- float
- long
- octet
- short
- string
- ulong
- ushort

All the other data types must be written as CORBA IDL type definitions. Complex types are created with struct, array and sequence keywords. To register defined types you can use the function:

```
void adr_type_init(type_repository **types);
```

8.4.7 IOM Support for Endpoints

The LCF supports the Inter-object Messaging (IOM) capability. It enables Endpoints to send bulk data "up" through the Endpoint gateway to other systems in the enterprise. The Endpoints that must spool a large amount of data back to the server, for example, can use IOM to do this.

IOM is efficient for distributing large amounts of data because it bypasses the object dispatcher. In the Lightweight Client Framework, the gateway serves as an IOM proxy; it receives the data from the Endpoint and handles the distribution to the final destination.

For scalability reasons, IOM should not be used as a convenience tool for arbitrary Endpoint communication. It should be used only to send large amounts of data from an Endpoint back up to other systems. The following functions provide IOM support for Endpoints.

`iom_open`	Opens the IOM channel
`iom_send`	Sends the stream of data over the channel

`iom_close` Closes the IOM channel

8.4.8 Functions for Launching Processes

Several functions have been added to launch processes on UNIX and PC platforms. These functions simplify process launch and give a unified API across all platforms for performing these types of functions as the table below shows.

Table 56. Functions for Launching Processes

Routine	Description
tiv_create_process	Starts a WIN32 process.
tiv_create_process_ui	Launches a GUI-based command on a desktop other than the default desktop of the current process.
tiv_io_create	Creates an array to pass to tiv_spawn as the argument representing the file handles for stdin, stdout, and stderr.
tiv_io_destroy	Frees a pointer allocated by a call to tiv_io_create.
tiv_spawn	Spawns a process.
tiv_spawn_ui	Launches a GUI-based command on a desktop other than the default desktop of the current process.
tiv_user_token_create	Allocates a pointer to a token that contains information used by tiv_spawn for launching processes in the context of another user. Note: Because `setuid` is not supported, the public version in `libmrt` has no effect.
tiv_user_token_destroy	Deallocates the token created by calling tiv_user_token_create. Note: Because `setuid` is not supported, the public version in `libmrt` has no effect.
tiv_wait	Waits for an asynchronously spawned process to return.

8.4.9 Miscellaneous Functions

The table below shows miscellaneous functions provided for the Endpoint applications.

Table 57. Miscellaneous Functions

Routine	Description
decrypt_data	Decrypts application data
encrypt_data	Encrypts application data
ep_stream_read	Reads an MDist stream.
ioch_recv	Receives bytes from the IO channel
mrt_set_method_exit_mode	Sets an attribute to create a condition (restart or reboot) when a method exits
mrt_machine_id	Returns the machine ID
mrt_test_dependency	Determines if a dependency to a method has been updated. For use by long-running methods such as daemons
nw_echo_command_to_console	NetWare only; sends a command line to the console
set_lang	Sets the language to use to bind messages on the gateway
still_alive	Reinitializes the timer on the gateway to keep the gateway from timing out during an upcall or downcall

8.5 The Common Porting Layer Runtime Library

The Common Porting Layer library (`libcpl`) provides functions that are not serviced or behave differently in some system platforms. In this section, we introduce them.

8.5.1 Binary Tree Search Functions

These routines service binary search trees. All comparisons are done with a user-supplied routine. This routine is called with two arguments, which are pointers to the elements to be compared. The returned value of this routine is an integer that depends on the result of comparison between the first and the second argument, for example less than, equal to or greater than. The comparison function does not need to compare every byte; so arbitrary data may be contained in the elements in addition to the values to be compared. These functions are as follows.

`cpl_tsearch`	It is used to build and access the tree. If there is a datum in the tree equal to the value pointed by the argument, the pointer assumes that datum is returned. Otherwise, the argument is inserted, and the pointer is returned. As only pointers are copied, the calling routine must store the data.
`cpl_tfind`	Searches the datum in the tree. The `cpl_tfind` returns the pointer to the datum or the NULL pointer if the datum is not found.
`cpl_tdelete`	Deletes the node from a binary search tree. The `cpl_tdelete` returns the pointer to the parent of the deleted node, or a NULL pointer if the node is not found.
`cpl_twalk`	Traverses a binary search tree. Any node in a tree may be used as the root for a walk below that node.

The pointers to the key and the root of the tree should be of type pointer-to-element, and cast to type pointer-to-character. Similarly, although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

8.5.2 Directory Entry Functions

These functions enable the caller to use these APIs for all platforms without platform specific implementation details. The functions are:

- `cpl_opendir`
- `cpl_readdir`
- `cpl_rewinddir`
- `cpl_closedir`
- `cpl_telldir`
- `cpl_seekdir`

8.5.3 UNIX get Functions

These are the UNIX system calls that are not provided on other platforms. The brief description of these functions is as follows.

`cpl_getcwd`	Returns the current working directory of the calling process.
`cpl_getenv`	Gets environment variables.

`cpl_putenv`	Puts environment variables.
`cpl_getopt`	Extracts the command line switches and their arguments from the command line.
`cpl_getpass`	Queries user for a password (string) from t he standard input. The characters entered by the user are not echoed.
`cpl_gettimeofday`	Returns the time zone.
`cpl_gethostname`	Returns the host name of the system.

8.5.4 printf, fclose, fopen, getc Functions

Since these functions may not exist on some platforms, the common layer furnished wrappers for them as follows.

- `cpl_fprintf`
- `cpl_fclose`
- `cpl_fopen`
- `cpl_getc`

8.5.5 Temporary File Functions

Not all platforms provide sufficient function in the area of creating and manipulating temporary files. These APIs work around problems in the native version of these functions. These APIs are:

`cpl_tmpfile`	Creates a temporary file and returns a pointer to that stream (like the ANSI/C function).
`cpl_tmpman`	Generates a temporary filename that can be used to open a temporary file without overwriting an existing file (like the ANSI/C function).
`cpl_tmpdir`	It is an additional function and enables the application to query the location of the system temporary directory on the specific platform.

8.5.6 Callback Functions

Two registration callback functions are provided as a part of the common porting layer.

- `cpl_register_print_callback`: This function registers a function for handling LCF output messages generated inside `libmrt`. By default, all `libmrt` messages are printed to stdout. However, on the platforms such as Windows and NetWare, it is more useful to have an application-specific way of displaying these messages. The `cpl_register_print_callback` function provides this.

- `cpl_register_thread_yield_callback`: This function allows an application to register a callback function that will be called periodically during long `libmrt` operations. For example, some LCF operations, such as waiting for a task to complete or for a communication event to occur, may require that a program wait inside a `libmrt` function for an extended period of time. This can be a problem in single-threaded applications that need to handle other events during the operation. For example, Windows applications need to handle OS events for moving and resizing a window.

8.5.7 Miscellaneous Functions

There are other scattered functions that are also included in the Common Porting Layer.

`cpl_fflush`	Flushes a stream.
`cpl_ltoa`	Converts a long integer from a binary representation to a string representation.
`cpl_THREADyield`	Yields a timeslice for Windows, Windows 95 or NetWare platforms.
`uname`	Enables callers to query for the basic information about the running platform.
`stat Macros`	These are often accompanied by a series of macros that provide a simple interface to obtain useful information about files. The following macros are defined:

- `S_ISDIR`: Queries if the file is a directory.
- `S_ISCHR`: Queries if the file is a character special device.
- `S_ISFIFO`: Queries if the file is a FIFO.
- `S_ISREG`: Queries if the file is regular.

`wstat Macros`	Provide information of the `wstat` macros for use on PC platforms.

8.6 Dependencies

An Endpoint method may be implemented across several files. Besides the executable binary, a method may also require supporting files, such as a shared library, message catalogs, or other software (for example, a Perl interpreter). This chapter discusses how the files supporting an Endpoint method are identified and downloaded from the Endpoint gateway to the Endpoint to be ready for the method invocations.

On the Managed Node, all binaries, libraries, message catalogs and other runtime files that a method requires are installed on the Managed Node at application installation time. The method implementation database only identifies the method body, the binary program, or script that contains the method entry point.

The Endpoints, however, do not have any methods or supporting files present on them when they are initially installed. Method bodies are identified from the method implementation database and downloaded from the gateway on demand. The files supporting the method must also be identified and downloaded from the gateway to the Endpoint.

When an Endpoint application is installed, the method bodies and supporting files (called dependencies) are copied to each gateway and are stored in the gateway repository, the $BINDIR/../lcf_bundle directory on the gateway. When the gateway is called to invoke an Endpoint method for the first time, it searches its repository for the method body and its dependencies, then downloads them to the Endpoint.

This section describes the commands to use for managing dependencies in the LCF environment and provides examples of usage for two commands, `wdepset` and `wchdep`.

8.6.1 The LCF Dependency Mechanism

The Lightweight Client Framework provides the dependency mechanism that supports downloading method bodies and their supporting files from the gateway to Endpoints. The dependency mechanism provides the ability to:

- Specify a set of dependencies for a method
- Associate those dependencies with that method
- Download those dependencies when the method needs them

When you install your application, you first create the dependency set (using the `wdepset` command) and then associate it with the method header (using the `wchdep` command).

8.6.2 Usage for Dependencies

While there are many uses for the dependency mechanism, the following are three general uses for dependencies.

- To download a dependency for use by a method in an application
- As a part of the product installation, to download utilities for users on the Endpoint
- To download tools (principally Windows NT tools) to the Endpoint for use by an Endpoint application f

This following sections explain how the method uses the dependency mechanism.

8.6.3 The Gateway Repository

The Endpoint gateway repository is located in the $BINDIR/../lcf_bundle directory on the Endpoint gateway. When the gateway downloads a file to an Endpoint, it searches the gateway repository to find the file. For example, if a method implementation is specified as LCF/test, then the gateway looks for the actual file in $BINDIR/../lcf_bundle/bin/$INTERP/LCF/test. When the gateway receives a method invocation on behalf of an Endpoint, the gateway checks its repository for the dependencies associated with each method. The dependencies are downloaded to the Endpoint at the same time as the method body.

Dependency information for methods is stored on the TMR server and is cached by the gateway. Dependency information consists of the names of the dependent files and their associated flags specified with the `wdepset` command.

When an Endpoint invokes a method, the Endpoint gateway and Endpoint "converse" to determine if a new version of the method body or any dependencies need to be downloaded. The Endpoint tells the gateway which method and dependency versions it has, and the gateway downloads new versions to the Endpoint if needed.

8.6.4 Location for Storing Dependencies on the Endpoint

When dependencies are downloaded from the gateway to the Endpoint, by default they are stored in the Endpoint's method cache. The Endpoint's method cache is the value of the `LCF_CACHEDIR` environment variable.

Dependencies stored in the Endpoint's method cache may be deleted if the cache is full and new dependencies are needed to be downloaded. The default size of the Endpoint method cache is 20.5 MB; it can be changed with the `-D cache_limit` option of the `lcfd` command.

Dependencies can also be downloaded to a directory other than the Endpoint's method cache. These are called out-of-cache dependencies. For an out-of-cache dependency, you must specify an Endpoint directory when you specify the dependency set.

In general, specify an out-of-cache location for a dependency such as a tool that needs to reside permanently on the Endpoint. Unlike dependencies stored in the Endpoint's cache directory, dependencies stored in an out-of-cache location are never deleted.

8.6.5 The DependencyMgr Object

Dependency information is stored on a `DependencyMgr` object, which honors the `Depends::Mgr` IDL interface. You specify this object when you use the `wchdep` command to associate a method with its dependency.

You can create a new class to hold your dependency information, but normally, your class will honor the `Depends::Mgr` interface.

8.6.6 Steps for Managing Dependencies

The remainder of this chapter discusses the following general steps for managing dependency sets when you install your product or patch:

1. Use the `wdepset` command to specify the dependency sets after the script.

   ```
   wdepset -c ds-depset -a bin LCF/test/upcall
   ```

2. Use the `wchdep` command to associate the dependency set with each method in the after script.

   ```
   wchdep @Classes:Downcall @DependencyMgr:ds-depset invoke
   ```

3. Use the `wgateway gateway_label dbcheck` command to synchronize the Endpoint gateway cache with the TMR server. You do this only once for all dependency sets, not for each dependency set.

   The gateway caches method invocation information when an Endpoint method is invoked. If an application is modifying the method, you must run the `wgateway gateway_label dbcheck` command for all gateways that service Endpoints.

   ```
   wgateway kodiak-gateway dbcheck
   ```

8.6.7 Using Dependencies to Deploy Tools to Endpoints

Tools needed by applications are included in the gateway repository. Applications (for example, the task library) that need access to tools such as the Bourne Shell (`sh.exe`) and Perl (`perl.exe`) on the Endpoint must establish the dependency for the `lcfd` to download the tool(s) on demand.

To establish this dependency, first create a dependency set (with the `wdepset` command) and then associate it with a method (with the `wchdep` command).

8.6.7.1 Tools Location

Tools are furnished for both UNIX and Windows NT interpreter types, though the tools are more likely to be used for Windows NT Endpoints.

For NT, the tools are located in the $BINDIR/../lcf_bundle/bin/w32-ix86/tools directory. It contains tools such as awk.exe, tail.exe, perl scripts are located in the lcf_bundle/bin/w32-ix86/tools/lib/perl directory.

For UNIX interpreter types, the tools are located in the $BINDIR/../lcf_bundle/bin/$INTERP/tools/bin directory. For example, for solaris2, the tools are located in the $BINDIR/../lcf_bundle/bin/solaris2/tools/bin directory. The Perl scripts are located in the lcf_bundle/bin/solaris2/tools/lib/perl directory.

8.6.7.2 Deploying Application Tools to an Endpoint

This example demonstrates how to use dependencies to deploy the tools (such as bash or Perl) to a Windows NT Endpoint on demand. Use the `wdepset` and `wchdep` commands to pull the tools down to the Endpoint.

1. Create the Dependency:

   ```
   wdepset -c task-library-tool-base \
   -a w32-ix86 bin/w32-ix86/tools/sh.exe +a +p %TOOLS% \
   -a w32-ix86 bin/w32-ix86/tools/win32gnu.dll +a +p %TOOLS%
   ```

2. Associate the dependency with a method. For example, the following associates the dependency with the `run_task` method on the `TaskEndpoint` object:

   ```
   wchdep @Classes:TaskEndpoint \
   @DependencyMgr:task-library-tool-base run_task
   ```

3. Synchronize the gateway’s method cache with that on the TMR server:

   ```
   wgateway kodiak-gateway dbcheck
   ```

where `kodiak-gateway` is the name of the Endpoint gateway in our test environment. Do this for each gateway.

8.7 TMA Sample Application

This section introduces a simple sample application for your understanding. The sample application illustrates the following:

- How to structure an LCF application into Endpoint code, platform code, and common code (shared by the Endpoint and platform).
- How to write a downcall program.
- How to write an upcall program.
- How to compile, link, and configure programs.

The sample application consists of the following modules.

- A command that is invoked on the gateway and makes downcalls (`dsmain`)
- A downcall method library that runs on the Endpoint (`downcall`) and invokes an upcall command (`upcall`)
- A command for upcall that runs on the Endpoint (`upcall`)
- An upcall method library that runs on the gateway (`upmeth`)

8.7.1 Process Sequence

The process sequence of the sample application is as follows.

8.7.1.1 Downcall Sequence

Figure 115 shows the downcall sequence.

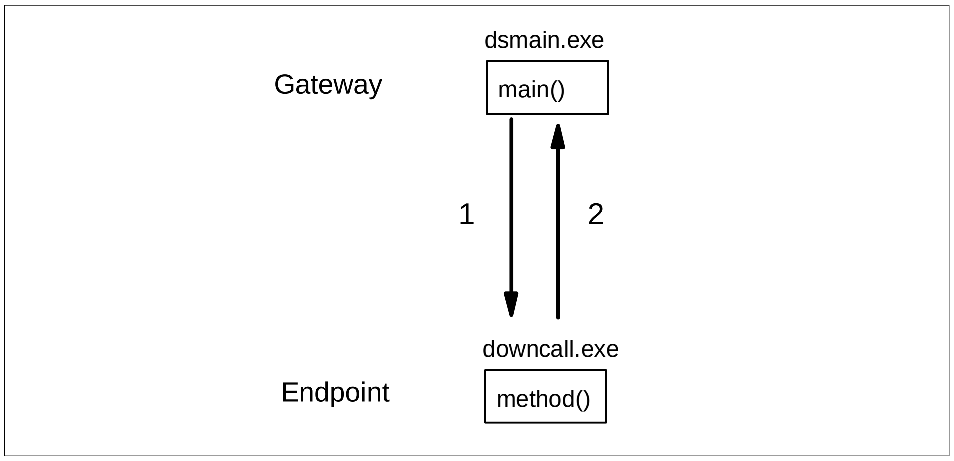


Figure 115. Downcall Sequence of a Sample Application

1. The downcall client, `dsmain`, calls the remote method, `method`, that is invoked as a downcall. `dsmain` sends one string parameter in this downcall. `method` is linked in the module downcall.exe.
2. The method, `method`, simply converts the input string to the uppercase and returns it to the caller.

8.7.1.2 Upcall Sequence

Figure 116 shows the upcall sequence.

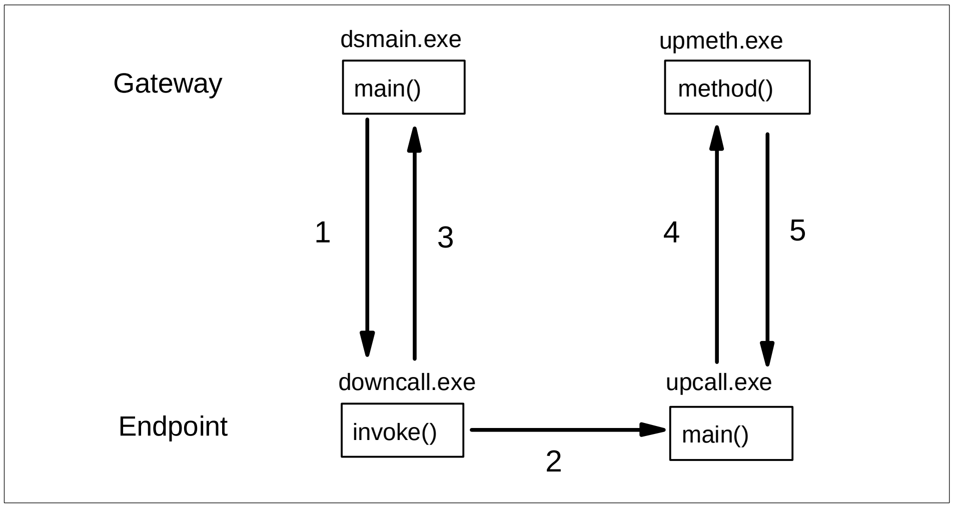


Figure 116. Upcall Sequence of a Sample Application

1. `dsmain` calls the remote method `invoke` that is linked in downcall.exe. Before calling `invoke`, the gateway downloads dependency upcall.exe to the Endpoint. Then the downcall will be invoked.
2. The method `invoke` spawns a new child process with the CLI module `upcall.exe`.
3. The `invoke` method returns the control to `dsmain` immediately without waiting for termination of the child process.
4. Upcall.exe calls the method, `method`, as an upcall. `method` is linked in the module upmeth.exe. `Upcall` sends one string parameter in this upcall.
5. The upcall server simply returns input string to the caller. Upcall.exe logs the returned message.

8.7.2 The Source Tree

This section discusses the directory and file structure for the sample application; use a similar structure for an application that has both platform and Endpoint components. The diagram below shows the structure of the source tree for the sample application:

```
/src
 ├──── /upsamp
 │       ├──── /common
 │       ├──── /endpoint
 │       │       └──/src
 │       └──── /platform
 │               └──/src
 └──── /downsamp
         ├──── /common
         ├──── /endpoint
         │       └──/src
         └──── /platform
                 └──/src
```

Figure 117. Source Tree

Each application has three directories:

Endpoint Contains files unique to the Endpoint side.

Platform Contains files unique to the server side.

Common Contains files common to both Endpoint and server. Note that no builds take place in this directory; there are no make files. The common files are referenced by relative path from the Endpoint and platform directories.

This directory tree has several important advantages:

- As you develop your application, it enables you to build and test the Endpoint and server sides independently of each other.
- It makes porting your application easier; that is, when you are porting your application to a new platform, you only have to port the Endpoint-side code.

If necessary, you can use `#ifdef ENDPOINT_BUILD` in your source files to distinguish between the TME 10 server-side and the Endpoint-side code. This is the standard way that Tivoli recommends.

8.7.3 Source Files

This section provides the sample source file information.

Note

Normally, we use Makefile to compile and link programs. But using Makefile is hard to understand for our entry-level discussion. So in this sample application, we do not use Makefile and compile and link step-by-step.

The following table lists the files for the sample application by directory (common, endpoint, and platform):

Table 58. Source Files

Application	Directory	Files
Upsamp	Common	`common/upcall.gen`
	Endpoint	`endpoint/src/upcall.c`
	Platform	`platform/src/upmeth.c`
		`platform/src/upcall.init`
Downsamp	Common	`common/downcall.gen`
	Endpoint	`endpoint/src/downcall.c`
	Platform	`platform/src/dsmain.c`
		`platform/src/downcall.init`
		`platform/src/dep`

8.7.3.1 Files in the Upsamp Directory

The following files are located in the directory used for the Upsamp application.

upcall.gen	The generation file for the upcall sample program. This file is used to generate idl/ist/imp/prog files automatically using the sgen utility.
upcall.c	The upcall client program that runs on the Endpoint.

upmeth.c	The upcall server program that runs on the gateway.
upcall.init	The initialization script for the upcall server object.

8.7.3.2 Files in the Downsamp Directory

The following files are located in the directory used for the Downsamp application.

downcall.gen	The generation file for the downcall sample program. This file is used to generate idl/ist/imp/prog files automatically using the sgen utility.
downcall.c	The downcall server program that runs on the Endpoint.
dsmain.c	The downcall client program that runs on the gateway.
downcall.init	The initialization script for the downcall server object.
dep	The script used to configure dependencies.

8.7.4 The Upsamp Files

This section briefly describes the contents of each file in the sample program, by directory.

8.7.4.1 The IDL Files

This section describe the IDL files that are common to both Endpoint and server sides:

- Upcall.gen
- Upcall.idl
- Upcall.imp
- Upcall.prog
- Upcall.ist

The Generation File: Upcall.gen

The file Upcall.gen defines the information to generate idl/imp/prog/ist files. For detail, refer to each file description in the following.

Note

The sgen utility is not documented officially; so if you do not understand all of these parameters, we do not recommend using it. You can create its outputs (.idl, .ist, .prog, .imp) manually.

```
/*
    Sample generation source for sgen utility

*/
fname= Upcall

#pragma generate False
#ifdef ENDPOINT_BUILD
#  include <mrt/stypes.idl>
#else
#  include <tivoli/TMF_Types.idl>
#endif
#pragma generate True

/* IDL module name */
module= LCF

    /* IDL interface name */
    interface= Upcall

    /* list of methods */
    method= method

        /* method definition */
        void method(in string input, out string output);

        /* server execution model */
        model=per method

        startup=
            "tas_init"

        interp=default

        path= /LCF/test/upmeth

        igoo=
            acldefault { "any" };
            initialize { "$METHODPATH/upcall.init $imp_LCF_iUpcall_CO \
                  $imp_LCF_iUpcall_BO";};
```

The IDL Definition File: Upcall.idl

The file Upcall.idl contains the following primary components.

- A module specification, `LCF`
- An interface, `Upcall`
- `Upcall` interface has an operation, `method`, supported by gateway
- The `method` operation has one input parameter, `input` and one output parameter, `output`.

This interface constitutes the gateway component of the application. The code for the Upcall.idl is as follows:

```
#if !defined Upcall_idl
#define Upcall_idl

#pragma generate False
#ifdef ENDPOINT_BUILD
#  include <mrt/stypes.idl>
#else
#  include <tivoli/TMF_Types.idl>
#endif
#pragma generate True

/* IDL module name */
module LCF {

    /* IDL interface name */
    interface Upcall {
        /* method definition */
        void method(in string input, out string output);

    };
};
#endif
```

The Implementation File: Upcall.imp

The TEIDL implementation specification is a block of information that specifies the implementation of a class of objects. The implementation file describes the class implementation, which includes the following:

- Whether the class is instantiable
- The class attributes
- The class methods

The file Upcall.imp contains the following primary components.

- A preprocessor directive. The preprocessor directive instructs the compiler to include the file that contains the interfaces associated with this implementation.
- A module specification, `imp_LCF`
- An implementation, `iUpcall`. It is an abstract class. It acts as the base class for inheritance and cannot be instantiated. It honors the `LCF::Upcall` interface.
- A method, `method`, has a method implementation.
- The language, ANSI C. It binds for the method.

The iUpcall class specification describes how the Upcall interface is to be implemented. The newly specified class will, for example, provide a method called method, which implements the method operation specified in the Upcall IDL interface. The code for Upcall.imp is shown below.

```
#if !defined Upcall_imp
#define Upcall_imp

#include "Upcall.idl"

module imp_LCF {

    implementation abstract class iUpcall honors LCF::Upcall {

        methods { {
            method
        } binding = ansi C; };
    };
};
#endif
```

The binding keyword specifies whether a method is implemented as a C program, a shell script, or a command.

ansi C	Specifies the implementation of a method as a C
shell	Specifies the implementation of a method as a shell script
command	Specifies that the method to be supplied with a special helper execution environment that allows the method to read from stdin and write to stdout and stderr

The Program File: Upcall.prog

The TEIDL program specification provides additional details about the implementation. The implementation file does not describe the operating system process characteristics of class methods. You can, for example, implement the class methods to start, run, and then to terminate. Alternatively, you can implement them so that they start and then continue to run without terminating.

Such implementation details are called the method execution style. The TEIDL program specification describes the execution style of the class methods. The file Upcall.prog specifies the following for the method method:

- A preprocessor directive
- A module specification, prog_LCF
- A program specification, Upcall_main
- Method name, method

- Server type. The `method` method is to be executed as a "per-method" method; that is, the server executes it and then terminates it.
- Start-up specification. When the server is first activated, the BOA executes the functions specified by startup. In this case, it specifies that when a call is made to method, the server checks to see if `tas_init` is running. If not, it launches a process to execute it.

The code for Upcall.prog is shown below:

```
#if !defined Upcall_prog
#define Upcall_prog

#include "Upcall.imp"

module prog_LCF {

    program Upcall_main for imp_LCF::iUpcall {

        executes {
            method
        } per method;
        startup {
            "tas_init"

        };
    };
};
#endif
```

The server type keyword specifies how the server process is executed.

`per method`	Specifies that the server process execute one method and then terminate
`daemon`	Specifies a server that is a daemon process
`external daemon`	Specifies a server that is a daemon process not activated by the ORB

The Installation File: Upcall.ist

When a client initiates a request, the ORB locates the appropriate object implementation and performs a security check. For this reason, you must address two issues regarding object installation: location and security. The TEIDL installation constructs provide a convenient way to address these two issues.

One set of installation constructs specifies the installation path to one or more programs. Another set specifies the required authorization for each operation. The Upcall.ist contains the following specifications.

- A preprocessor directive.
- A module specification, `ist_LCF`.
- An installation specification, `istUpcall` for the implementation, `imp_LCF::iUpcall` with program `prog_LCF::Upcall_main`.
- The `acldefault` keyword specifies the default ACL during the installation, in this case, it is `any`.
- The `initialize` specifies a user-defined program name and its arguments to initialize a newly created object. The compiler-generated install script Upcall.cfg calls this program to initialize an object after the object is created.
- The `external path` specifies the path location as a disk file relative to the TME 10 bin directory; can be used with the default keyword.

  The code for Upcall.ist is shown below:

```
#if !defined Upcall_ist
#define Upcall_ist

#include "Upcall.prog"

module ist_LCF {

    installation istUpcall for imp_LCF::iUpcall with prog_LCF::Upcall_main {

            acldefault { "any" };
            initialize { "$METHODPATH/upcall.init $imp_LCF_iUpcall_CO \
                            $imp_LCF_iUpcall_BO";};
        external path { {
        method
        } = {
            "default", "/LCF/test/upmeth";
            };
        };
    };
};
#endif
```

Installation Preprocessor Directive

The include directive instructs the compiler to include the Upcall.prog program file, which completes the required file references for the `iUpcall` implementation. The following scenario explains the typical compile scheme for the sample implementation:

1. The installation file is specified as the compiler input using the following CLI command:

   ```
   tidlc Upcall.ist
   ```

2. The compiler compiles the installation file as instructed.
3. The include directive in the installation file instructs the compiler to include the program file (Upcall.prog), and the compiler also compiles the program file.
4. The include directive in the program file instructs the compiler to include the implementation file (Upcall.imp), and the compiler also compiles this file as instructed.
5. The include directive in the implementation file instructs the compiler to include the interface definition file (Upcall.idl), and the compiler also compiles this file. If there are any include directives in the interface definition file, the compiler includes and optionally compiles those files. Whether the files are compiled or not depends on the usage of the:

   ```
   #pragma generate
   ```
6. The compiler generates (among other things) client files which present the framework for the application.

   It is not a mandatory process, but using the described include scheme is recommended for simplicity.

8.7.4.2 Endpoint File

This section describes the method for the Endpoint side of the upcall sample.

Upcall.c

Upcall.c contains the implementation for the upcall CLI. It calls the stub, `t_LCF_Upcall_method`, and logs some records to the /temp/upcall.log file.

```
/*
    upcall.c

    Sample upcall client program

*/
#ifdef ENDPOINT_BUILD    /* Should be defined */
#include <mrt/tiv_mrt.h>
#endif
#include "t_Upcall.h"
#include "Upcall_aux.h"

int loglevel = 2;
char *upcall_log = "/temp/upcall.log";

int main(int argc, char *argv[])
{
    int error = 0;
    lh_p_t uplh = (lh_p_t)NULL;

    LogQueueAlloc(LOGQUEUEDEFSZ);
    if((uplh = LogInit(upcall_log,"upcall", loglevel,FALSE, LOGDEFSZ)) == NULL) {
        ++error;
    }
    LogSetDefault(uplh);
    LogMsg(1, LOGDEF, genKey(hello), "Starting Upcall");
    Try {
        Environment ev_dat;
        char * input_key = (argc > 1) ? argv[1] : NULL;
        char * out_string;
    mrt_set_upcall_timeout(60);/* set timeout */
    /* Do upcall */
        t_LCF_Upcall_method("Upcall", &ev_dat, 0,
            input_key, &out_string);
        if (out_string)
            LogMsg(1, LOGDEF, nullKey(),
                "out_string=\"%1$s\"", out_string);
        mg_free(out_string);/* free returned area */

    Catch(Exception, ex) {
        char *m = def_ex_bind(ex);
        LogMsg(1, LOGDEF, nullKey(),
                "Upcall test exception: %1$s\n", m);
        mg_free(m);
        ++error;
    }
    EndTry;
    LogMsg(1, LOGDEF, genKey(goodbye), "Ending Upcall");
    LogQueueDealloc();
    LogDeinit(uplh);
    exit(error);
}
```

8.7.4.3 Platform File

This section describes the other files regarding the upcall sample program.

Upmeth.c

The Upmeth.c implements the upcall method that runs on the gateway. The following shows the code for Upmeth.c.

```
/*

    upmeth.c

    Sample upcall server program

*/
#include "t_Upcall.h"
#include <tivoli/ExException.h>

void t_imp_LCF_iUpcall_method(
    LCF_Upcall _LCF_Upcall,
    Environment * _ev,
    transaction _transaction,
    char *input,
    char **output)
{
/* Return message */
    if (input && input[0]) {
        *output = mg_malloc(64000);
        sprintf(*output, "Message from gateway[%s]",input);
    }
/* If input does not exist, throw exception */
    else {
        Throw(ExSystem_new("Sample Exception from upcall server"));
    }
}
```

Upcall.init

The Upcall.init is the initialization file called from the TEIDL compiler-generated configuration script, Upcall.cfg. It sets the class-friendly name and enables the class.

The first argument passed to the program is the object reference of the newly created object. This reference is produced by the configuration script in the form of a Bourne shell variable. The variable takes the form: `$module_impl_CO`. `module` presents the module if one exists. The `impl` means implementation (class) name. `CO` indicates the variable is a class object reference. Additional arguments are passed to the program exactly as specified as the initialize keyword in the installation specification file (.ist). The following shows code for the Upcall.init.

```
#!/bin/sh -xe
#
# Initialization script for sample upcall server
#
if [ $# -gt 1 ]; then
    upsamp_CO=$1
    upsamp_BO=$2
else
    upsamp_CO=$imp_LCF_iUpcall_CO
    upsamp_BO=$imp_LCF_iUpcall_BO
fi

# Class friendly name
set_class_name_cli $upsamp_CO "Upcall"

objcall $upsamp_BO om_enable method TRUE TRUE
```

8.7.5 The Downsamp Files

This section briefly introduces the contents of each file in the sample program.

8.7.5.1 The IDL Files

This section describes about the IDL files that are common to both that Endpoint and server sides:

- Downcall.gen
- Downcall.idl
- Downcall.imp
- Downcall.prog
- Downcall.ist

The Generation File: Downcall.gen

The file Downcall.gen is similar to the file Upcall.gen. The content of the file is shown below.

```
/*
    Sample generation source for sgen utility

*/
fname= Downcall

#pragma generate False
#ifdef ENDPOINT_BUILD
#  include <mrt/stypes.idl>
#else
#  include <tivoli/TMF_Types.idl>
#endif
#pragma generate True

/* IDL module name */
module= LCF

    /* IDL interface name */
    interface= Downcall

    /* list of methods */
        method= method,invoke

        /* method definitions */
        void method(in string input, out string output);
        void invoke(in string input);

        /* server execution model */
        model=per method

        startup=

        interp= default

        path=/LCF/test/Downcall

        igoo=
            acldefault { "any" };
            initialize { "$METHODPATH/downcall.init $imp_LCF_iDowncall_CO \
                $imp_LCF_iDowncall_BO";};
```

The IDL Definition File: Downcall.idl

The file Downcall.idl defines a module, LCF, and specifies one interface, `Downcall`. It specifies two operation, method and invoke, supported by the Endpoint. The method operation has one input parameter, `input` and one output parameter, `output`. The `invoke` operation has one input parameter. This interface constitutes the Endpoint component of the application. The code for Downcall.idl is shown below:

```
#if !defined Downcall_idl
#define Downcall_idl

#pragma generate False
#ifdef ENDPOINT_BUILD
#  include <mrt/stypes.idl>
#else
#  include <tivoli/TMF_Types.idl>
#endif
#pragma generate True

/* IDL module name */
module LCF {

    /* IDL interface name */
    interface Downcall {

        /* method definitions */
        void method(in string input, out string output);
        void invoke(in string input);
    };
};
#endif
```

The Implementation File: Downcall.imp

The file Downcall.imp is similar to Upcall.imp. The code for Downcall.imp is shown below:

```
#if !defined Downcall_imp
#define Downcall_imp

#include "Downcall.idl"

module imp_LCF {

    implementation abstract class iDowncall honors LCF::Downcall {

        methods { {
            method,invoke
        } binding = ansi C; };
    };
};
#endif
```

The Program File: Downcall.prog

The program file specifies the execution characteristics for methods in a particular class. In this case, Downcall.prog specifies that the method `method` and `invoke` to be executed as a "per-method" method. The code for Downcall.prog is shown below:

```
#if !defined Downcall_prog
#define Downcall_prog

#include "Downcall.imp"

module prog_LCF {

    program Downcall_main for imp_LCF::iDowncall {

        executes {
            method,invoke
        } per method;
    };
};
#endif
```

The Installation File: Downcall.ist

The Downcall.ist is similar to Upcall.ist. The code for Downcall.ist is shown below:

```
#if !defined Downcall_ist
#define Downcall_ist

#include "Downcall.prog"

module ist_LCF {

    installation istDowncall for imp_LCF::iDowncall with prog_LCF::Downcall_main {

            acldefault { "any" };
            initialize { "$METHODPATH/downcall.init $imp_LCF_iDowncall_CO \
                $imp_LCF_iDowncall_BO";};
        external path { {
            method,invoke
        } = {
            "default", "/LCF/test/Downcall";
            };
        };
    };
};
#endif
```

8.7.5.2 Endpoint File

This section describes the method for the Endpoint side of the downcall sample.

Downcall.c

The Downcall.c contains the implementation for the downcall methods `method` and `invoke` that runs on the Endpoint.

```
/*
    downcall.c

    Sample downcall server program

*/
#include <mrt/tiv_mrt.h>
#include <mrt/sub_proc.h>
#include <mrt/mrtwrap.h>
#include <mrt/mrt_run.h>
#include "t_Downcall.h"

void t_imp_LCF_iDowncall_method(
    LCF_Downcall _LCF_Downcall,
    Environment * _ev,
    transaction _transaction,
    char *input,
    char **output)
{
    LogQ("Entering downcall method");

    if (input != 0 && input[0] != 0) {
        char *p = *output = ml_ex_malloc(strlen(input) + 1);
        memcpy(*output,input,strlen(input) + 1);
        // convert to uppercase and return
        _strupr(*output);
    }
    else {
        vaThrow(CatSpec(none, unknown1), "Exception: unknown input.");
    }
}

void t_imp_LCF_iDowncall_invoke(
    LCF_Downcall _LCF_Downcall,
    Environment * _ev,
    transaction _transaction,
    char *input)
{
    char tmpbuf[255];
    char * argv[3];
    void * ioptr;
    int pid,status;
    TIV_USER_T tuser = 0;
    argv[0] = tmpbuf;
    argv[1] = input;
    argv[2] = 0;
    sprintf(tmpbuf,"%s/cache/bin/w32-ix86/LCF/test/upcall.exe",
        mrt_expand_variable("RUN"));
    LogQ("Entering downcall method for invoking upcall");
    ioptr = tiv_io_create(IO_UNUSED,NULL,IO_UNUSED,NULL,IO_UNUSED,NULL);
    if (pid = tiv_spawn(0,argv,0,ioptr,0) == -1)
        LogQ("spawn error");
    LogQ("Exit downcall method for invoking upcall");
}
```

8.7.5.3 Platform File

This section describes the other files related to the downcall sample program.

Dsmain.c

The *Dsmain.c* implements the downcall CLI that runs on the Server. Dsmain calls stub `t_LCF_Downcall_method` and `t_LCF_Downcall_invoke`. The following shows the code for the Dsmain.c.

```
/*
    dsmain.c

    Sample downcall client program

*/
#include <stdio.h>
#include <tivoli/ExException.h>
#include <tivoli/tas_init.h>
#include <tivoli/dir.h>
#include <tivoli/tmfthr.h>

#include "t_Downcall.h"
#include "Downcall_aux.h"

int errors = 0;
char msgbuf[2048];

void usage(char *prog)
{
    fprintf(stderr, "Usage: %s Endpoint Message\n", prog);
    exit(1);
}

void do_downcall(char * oid, char * input_string)
{
    Environment ev_dat;
    char * out_string;

    printf("Making downcall with %s\n",input_string);
    t_LCF_Downcall_method(oid, &ev_dat, 0, input_string, &out_string);
    if (out_string) {
        printf("[%s]", out_string);
        ORBfree(&out_string);
    }
}

void do_upcall(char * oid,char * input)
{
    Environment ev_dat;

    printf("Making downcall for invoking upcall\n");
    t_LCF_Downcall_invoke(oid, &ev_dat, 0,input);
}
```

```
char * build_endpoint_object(const char *epname) {
    char *cp, *cp2;
    char oid[500];
    Object endpoint_object, class_object, behavior_object;
    Environment ev;

    endpoint_object = dir_lookup_instance ("Endpoint", epname);
    class_object = dir_lookup_instance ("Classes", "Downcall");
    behavior_object = t_TMF_Root_MetaBase_get_behavior (class_object, &ev, Trans_none);

    /* now build up real oid */
    cp = strchr(strchr(endpoint_object, '.') + 1, '.') + 1;
    *cp = 0;
    cp = strchr(strchr(behavior_object, '.') + 1, '.') + 1;
    /* get rid of # stuff if there */
    if ((cp2 = strchr(cp, '#')) != 0) cp2 = 0;
    sprintf(oid, "%s%s+", endpoint_object, cp);
    mg_free(endpoint_object);
    mg_free(class_object);
    mg_free(behavior_object);
    return mg_strdup(oid);
}

void main(int argc, char *argv[])
{
    char * oid;
    type_repository *t[1];

    if (argc < 2) {
        usage(argv[0]);
        exit(1);
    }

    t[0] = type_repository_null;

    tmf_init(t);    /* Initialize CORBA runtime */
    tmf_client_init();    /* Declare me as a client */
    tas_init();   /* Initialize TAS library (log,...) */
/* invoke normal downcall */
    Try {
        oid = build_endpoint_object(argv[1]);
        do_downcall( oid,argv[2]);
    }
    Catch(Exception, ex) {
        fprintf(stderr, "Downcall Exception: %s\n",
            tmf_ex_msg_bind(ex,msgbuf,sizeof(msgbuf)));
        errors++;
    }
    EndTry;
/* invoke downcall to invoke upcall on the Endpoint */
    Try {
        do_upcall(oid,argv[2]);
    }
    Catch(Exception, ex) {
        fprintf(stderr, "Downcall Exception: %s\n",
            tmf_ex_msg_bind(ex,msgbuf,sizeof(msgbuf)));
        errors++;
    }
    EndTry;

    exit(errors);
}
```

Downcall.init

The Downcall.init is the initialization file called from the TEIDL compiler-generated configuration script, Downcall.cfg. It sets the class-friendly name. The following shows code for the Downcall.init.

```
#!/bin/sh -xe
#
# Initialization script for sample downcall server
#
if [ $# -gt 1 ]; then
    downcall_CO=$1
    downcall_BO=$2
else
    downcall_CO=$imp_LCF_iDowncall_CO
    downcall_BO=$imp_LCF_iDowncall_BO
fi

# Class friendly name
set_class_name_cli $downcall_CO "Downcall"

echo "LCF downcall test behavior object is $downcall_BO"
```

dep

The dep is the configuration script that defines dependency. This script defines dependency set `ds-depset` and creates a relationship with `Downcall::invoke`. This means the upcall CLI will be downloaded by the gateway before executing the `Downcall::invoke`.

```
#!/bin/sh
# script to configure dependency
#
APP_LABEL=ds-depset
if wlookup -r DependencyMgr $APP_LABEL ; then
    wdepset -d @DependencyMgr:$APP_LABEL
fi
wdepset -c $APP_LABEL -a bin LCF/test/upcall
wchdep @Classes:Downcall @DependencyMgr:$APP_LABEL invoke
for gw in ‘wlookup -L -r  gateway -a‘
do
    set +e
    wgateway $gw dbcheck
    set -e
done
exit 0
```

8.7.6 The Export Tree

When the build completes, you find the following subdirectories in both the platform and Endpoint directories:

export	Directory where the entire build is merged. This directory serves as a collection point for files for all interpreter types, created during the build.
interp	One interp directory for each platform you build on, created during the build.
src	Directory where you checked out the source to build.

All targets should be copied from the build tree to the export directory. Both the Endpoint side and the server side contain an export directory. The figure below shows the structure of the export directory for the sample application.

```
/src
 └──── /export
          ├──── /bin
          │        └─── /interp
          ├──── /generic
          ├──── /cfg
          │        └─── /interp
          └──── /include
                   └─── /interp
```

Figure 118. The Export Tree

- The bin directory contains the application binaries. In this case, on the server side it, contains the executable `dsmain` and `upmeth`, while on the Endpoint side, it contains the executable `downcall` and `upcall`.
- The cfg directory contains the scripts and configuration files necessary to install the application's classes. In this case, it contains the files Upcall.cfg, Upcall_ir.tar, Upcall_ist.tar, Downcall.cfg, Downcall_ir.tar, and Downcall_ist.tar.

  This directory exists only on the server side, not the Endpoint side.
- The include directory contains the application's public header files. This directory exists only on the server side, not on the Endpoint side.
- The generic directory includes shell scripts for general use of the application.

8.8 Building the Sample Application

To design the LCF application and write the IDL files, it is important to understand the components the application requires. For example, for sample application, there are two separate interfaces in the .idl file:

- The `Upcall` is the sample interface for the upcall.
- The `Downcall` is the sample interface for the downcall.

Each interface has its own implementation: two different sets of binaries for the distribution to two different systems. The LCF development process is different from the full Framework development process, which does not need to know the environment. The sample application illustrates the following basic steps for writing and building new LCF application.

1. Write the .idl, .imp, .prog, and .ist files for the application to provide interfaces for each component (server, gateway, Endpoint) of the application. These files are common to both the Endpoint and server side of the application and normally reside in the src/app/common directory.

   In this sample application, the `sgen` utility is used to generate those files automatically and the common directory contains the generated files only. Generated files will be located in the interp directory.

   Determine the parts exclusive to the Endpoint and to each platform and be sure those files are in their respective directories. In addition, determine the parts of the code in the application that are common to both the Endpoint and the platform.

   In this sample application, for example, dsmain.c is exclusive to the server side, and downcall.c is exclusive to the Endpoint side. The Upcall.c is exclusive to the Endpoint side, and upmeth.c is exclusive to the gateway.

2. Build the files on the server side by running the TEIDL compiler (`tidlc`) in the platform directory. Be sure to include the common files when you build them.

3. The first time you build the server side, the TEIDL compiler produces a stubbed-out version of the methods, among many other files. For example, in the sample application, the file t_Upcall_meth.c (the method template produced by the TEIDL compiler) is copied from the platform/interp directory to the Endpoint source directory, renaming its file name to something meaningful. In this case, it is named upmeth.c. This file contains the C code for the method, `method`. You should complete this source code to perform your application process.

4. Build the Endpoint side, using `ltid`. Notice that you are compiling the IDL files twice - once for the server side, using `tidlc`, and once for the Endpoint side, using `ltid`. Building LCF applications differs somewhat from full Framework builds. You build on both the Endpoint and server sides, rather than in one place. The common files are generated on both sides (Endpoint and platform), rather than being referenced from the common area. This is because there are two slight differences in the application development environments. The two sides also build slightly differently; the server side uses `tidlc`, and the Endpoint side uses `ltid`.

8.8.1 Sequence of Steps for Building a TMA Application

This section provides a step-by-step approach to building the sample application. We do not use Makefile and use only one operating system. We suppose that the platform and Endpoint will be run on Windows NT and using Microsoft Visual C++ 5.0 to compile and link the application. Also, we use the following installation directory in this section.

Note

The installation location of the Microsoft Visual C++ and Developer Studio should not contain blank character such as "Program Files\DevStudio". Although the default location may be C:\Program Files\DevStudio, you must use another location, such as C:\DevStudio.

Microsoft Developer Studio	C:\DevStudio
Microsoft Visual C++	C:\DevStudio\VC
Tivoli	C:\Tivoli
Endpoint	C:\Tivoli\lcf
Source file	C:\Tivoli\src\upsamp, C:\Tivoli\src\downsamp
Export file	C:\Tivoli\src\export

8.8.1.1 Environment Setup

To set the environment, we performed the following command.

1. Set up the compiler environment

   C:\DevStudio\vc\bin\vcvars32

2. Start bash

   ```
   sh
   ```

3. Set up the Tivoli Environment

   ```
   . /winnt/system32/drivers/etc/Tivoli/setup_env.sh
   ```

8.8.1.2 Building Upcall Server(Platform)

The following describes how to build the sample application (Upcall server).

1. Create the work directory.

   ```
   mkdir /Tivoli/src/upsamp/platform/w32-ix86
   ```

2. Create .idl, .imp, .prog, .ist files using the sgen utility.

   ```
   cd /Tivoli/src/upsamp/common
   sgen < Upcall.gen
   ```

3. Compile IDL inputs.

   ```
   cd /Tivoli/src/upsamp/platform/w32-ix86
   tidlc  -I. \
   ```

```
-I/Tivoli/include/w32-ix86 \
-I/Tivoli/include/generic \
../../common/Upcall.ist
```

4. Copy generated headers to the export directory.

```
mkdir -p /Tivoli/src/export/include/w32-ix86/tivoli
install.sh -m 0644 upcall_defs.h \
  /Tivoli/src/export/include/w32-ix86/tivoli/upcall_defs.h
install.sh -m 0644 upcall.h \
  /Tivoli/src/export/include/w32-ix86/tivoli/upcall.h
install.sh -m 0644 t_upcall.h \
  /Tivoli/src/export/include/w32-ix86/tivoli/t_upcall.h
install.sh -m 0644 upcall_aux.h \
  /Tivoli/src/export/include/w32-ix86/tivoli/upcall_aux.h
install.sh -m 0644 upcall_imp.h \
  /Tivoli/src/export/include/w32-ix86/tivoli/upcall_imp.h
install.sh -m 0644 t_upcall_imp.h \
  /Tivoli/src/export/include/w32-ix86/tivoli/t_upcall_imp.h
```

5. Compile sources.

```
export CCOPT="-MD -DWIN32 DSTDC_SRC_COMPATIBLE \
  -I. \
  -I/Tivoli/src/export/include/w32-ix86 \
  -I/Tivoli/src/export/include/generic  \
  -I/Tivoli/src/include/w32-ix86 \
  -I/Tivoli/src/include/generic"
cl $CCOPT -c ../src/upmeth.c -Foupmeth.obj
cl $CCOPT -c upcall_main.c -Foupcall_main.obj
cl $CCOPT -c upcall_main_skel.c -Foupcall_main_skel.obj
cl $CCOPT -c upcall_aux.c -Foupcall_aux.obj
cl $CCOPT -c upcall_imp.c -Foupcall_imp.obj
cl $CCOPT -c t_upcall_imp.c -Fot_upcall_imp.obj
cl $CCOPT -c t_upcall_main_skel.c -Fot_upcall_main_skel.obj
```

6. Link objects.

```
export LKOPT="/LIBPATH:/Tivoli/lib/w32-ix86
  -subsystem:console
  libtas.a libtmfimp.a libtmf.a libms.a
  libdes.a libthreads.a"
link $LKOPT \
  -out:upmeth.exe upmeth.obj \
  upcall_aux.obj \
  upcall_imp.obj \
  upcall_main.obj \
  t_upcall_imp.obj \
  upcall_main_skel.obj \
  t_upcall_main_skel.obj
```

7. Copy objects to the export directory.

```
mkdir -p /Tivoli/src/export/bin/w32-ix86/LCF/test
install.sh -m 0644 upmeth.exe \
  /Tivoli/src/export/bin/w32-ix86/LCF/test/upmeth.exe
install.sh -m 0644 upcall.cfg
/Tivoli/src/export/cfg/w32-ix86/tivoli/upcall.cfg
install.sh -m 0644 upcall_ir.tar \
  /Tivoli/src/export/cfg/w32-ix86/tivoli/upcall_ir.tar
install.sh -m 0644 upcall_ist.tar \
  /Tivoli/src/export/cfg/w32-ix86/tivoli/upcall_ist.tar
install.sh -m 0644 ../src/upcall.init \
  /Tivoli/src/export/cfg/w32-ix86/tivoli/upcall.init
```

8.8.1.3 Building Upcall Client (Endpoint)

The following describes how to build the sample application (Upcall client).

1. Create the work directory.

```
mkdir /Tivoli/src/upsamp/endpoint/w32-ix86
mkdir /Tivoli/src/upsamp/endpoint/cross
```

2. Compile the IDL inputs.

```
cd /Tivoli/src/upsamp/platform/cross
export PATH="c:/tivoli/bin/lcf_bundle/bin/w32-ix86/ade;$PATH"
/Tivoli/bin/lcf_bundle/bin/w32-ix86/ade/ltid -DENDPOINT_BUILD \
  -I. \
  -I/Tivoli/src/export/include/w32-ix86 \
  -I/Tivoli/src/export/include/generic  \
  -I/Tivoli/bin/lcf_bundle/include/w32-ix86 \
  -p ../../common/upcall.ist
```

3. Compile the source.

```
export CCOPT="-MD -DWIN32 -DENDPOINT_BUILD -DSTDC_SRC_COMPATIBLE -DPC \
  -I. \
  -I../cross/. \
  -I/Tivoli/bin/lcf_bundle/include/w32-ix86 \
  -I/Tivoli/bin/lcf_bundle/include/generic \
  -I/Tivoli/include/w32-ix86"

cd ../w32-ix86
cl $CCOPT -c ../src/upcall.c -Foupcall.obj
cl $CCOPT -c ../cross/upcall_aux.c -Foupcall_aux.obj
cl $CCOPT -c ../cross/t_upcall_stub.c -Fot_upcall_stub.obj
```

4. Link the objects.

```
export LKOPT="/LIBPATH:/Tivoli/bin/lcf_bundle/lib/w32-ix86 \
  -subsystem:console libmrt.a libdes.a libcpl.a"
```

```
link $LKOPT -out:upcall.exe upcall.obj \
  upcall_aux.obj \
  t_upcall_stub
```

5. Copy the objects to the export directory.

```
cp upcall.exe upcall
install.sh -m 0644 upcall \
/Tivoli/export/bin/lcf_bundle/bin/w32-ix86/LCF/test/upcall
```

8.8.1.4 Upcall Configuration

To invoke the sample program, we performed the following steps.

1. Copy the configuration files to the repository.

```
mkdir -p /Tivoli/bin/w32-ix86/LCF/test
cp /Tivoli/src/export/cfg/w32-ix86/Tivoli/upcall.* \
  /Tivoli/bin/w32-ix86/LCF/test
cp /Tivoli/src/export/cfg/w32-ix86/Tivoli/upcall_*.* \
  /Tivoli/bin/w32-ix86/LCF/test
```

2. Invoke the Configuration Script

```
/Tivoli/src/export/cfg/w32-ix86/Tivoli/upcall.cfg \
  test /Tivoli/bin w32-ix86 LCF
```

Note

The .cfg script can be invoked once. If you want to re-execute the .cfg script, you should:

1. Unregister the `Upcall` class

```
wregister -u -r Classes Upcall
```

2. Modify the .cfg file. Find the `create_class_cli` string and change it to the `update_class_cli` string.

8.8.1.5 Building the Downcall Client (Platform)

The following introduces how to build the sample application (Downcall client).

1. Create the work directory.

```
mkdir /Tivoli/src/downsamp/platform/w32-ix86
```

2. Create .idl, .imp, .prog, .ist files using the `sgen` utility.

```
cd /Tivoli/src/downsamp/common
sgen < Downcall.gen
```

3. Compile the IDL inputs.

```
cd /Tivoli/src/downsamp/platform/w32-ix86
tidlc  -I. \
-I/Tivoli/include/w32-ix86 \
-I/Tivoli/include/generic \
../../common/Downcall.ist
```

4. Copy the generated headers to the export directory.

```
mkdir -p /Tivoli/src/export/include/w32-ix86/tivoli
install.sh -m 0644 downcall_defs.h \
  /Tivoli/src/export/include/w32-ix86/tivoli/downcall_defs.h
install.sh -m 0644 downcall.h \
  /Tivoli/src/export/include/w32-ix86/tivoli/downcall.h
install.sh -m 0644 t_downcall.h \
  /Tivoli/src/export/include/w32-ix86/tivoli/t_downcall.h
install.sh -m 0644 downcall_aux.h \
  /Tivoli/src/export/include/w32-ix86/tivoli/downcall_aux.h
```

5. Compile the sources.

```
export CCOPT="-MD -DWIN32 DSTDC_SRC_COMPATIBLE \
  -I. \
  -I/Tivoli/src/export/include/w32-ix86 \
  -I/Tivoli/src/export/include/generic  \
  -I/Tivoli/src/include/w32-ix86 \
  -I/Tivoli/src/include/generic"
cl $CCOPT -c ../src/dsmain.c -Fodsmain.obj
cl $CCOPT -c downcall_aux.c -Fodowncall_aux.obj
cl $CCOPT -c t_downcall_stub.c -Fot_downcall_stub.obj
```

6. Link the objects.

```
export LKOPT="/LIBPATH:/Tivoli/lib/w32-ix86
  -subsystem:console
  libtas.a libtmfimp.a libtmf.a libms.a
  libdes.a libthreads.a"
link $LKOPT \
  -out:dsmain.exe dsmain.obj \
  downcall_aux.obj \
  t_upcall_stub.obj \
```

7. Copy the objects to the export directory.

```
install.sh -m 0644 dsmain.exe \
  /Tivoli/src/export/bin/w32-ix86/bin/dsmain.exe
install.sh -m 0644 downcall.cfg
/Tivoli/src/export/cfg/w32-ix86/tivoli/downcall.cfg
install.sh -m 0644 downcall_ir.tar \
  /Tivoli/src/export/cfg/w32-ix86/tivoli/downcall_ir.tar
install.sh -m 0644 downcall_ist.tar \
  /Tivoli/src/export/cfg/w32-ix86/tivoli/downcall_ist.tar
```

```
install.sh -m 0644 ../src/downcall.init \
  /Tivoli/src/export/cfg/w32-ix86/tivoli/downcall.init
```

8.8.1.6 Building the Downcall Server (Endpoint)

The following describes how to build the sample application (Downcall server).

1. Create the work directory.

```
mkdir /Tivoli/src/downsamp/endpoint/w32-ix86
mkdir /Tivoli/src/downsamp/endpoint/cross
```

2. Compile the IDL inputs.

```
cd /Tivoli/src/downsamp/platform/cross
export PATH="c:/tivoli/bin/lcf_bundle/bin/w32-ix86/ade;$PATH"
/Tivoli/bin/lcf_bundle/bin/w32-ix86/ade/ltid -DENDPOINT_BUILD \
  -I. \
  -I/Tivoli/src/export/include/w32-ix86 \
  -I/Tivoli/src/export/include/generic  \
  -I/Tivoli/bin/lcf_bundle/include/w32-ix86 \
  -p ../../common/downcall.ist
```

3. Compile the source.

```
export CCOPT="-MD -DWIN32 -DENDPOINT_BUILD -DSTDC_SRC_COMPATIBLE -DPC \
  -I. \
  -I../cross/. \
  -I/Tivoli/bin/lcf_bundle/include/w32-ix86 \
  -I/Tivoli/bin/lcf_bundle/include/generic \
  -I/Tivoli/include/w32-ix86"

cd ../w32-ix86
cl $CCOPT -c ../src/downcall.c -Fodowncall.obj
cl $CCOPT -c ../cross/downcall_aux.c -Fodowncall_aux.obj
cl $CCOPT -c ../cross/downcall_main_skel.c -Fodowncall_main_skel.obj
cl $CCOPT -c ../cross/downcall_main.c -Fodowncall_main.obj
```

4. Link the objects.

```
export LKOPT="/LIBPATH:/Tivoli/bin/lcf_bundle/lib/w32-ix86 \
  -subsystem:console libmrt.a libdes.a libcpl.a"

link $LKOPT -out:downcall.exe downcall.obj \
  downcall_aux.obj \
  downcall_main_skel.obj \
  downcall_main.obj
```

5. Copy the objects to export directory.

```
cp downcall.exe downcall
```

```
install.sh -m 0644 downcall \
/Tivoli/export/bin/lcf_bundle/bin/w32-ix86/LCF/test/downcall
```

8.8.1.7 Downcall Configuration

To invoke the sample program, we performed the following process.

1. Copy the configuration files to the repository.

```
cp /Tivoli/src/export/cfg/w32-ix86/Tivoli/downcall.* \
  /Tivoli/bin/w32-ix86/LCF/test
cp /Tivoli/src/export/cfg/w32-ix86/Tivoli/downcall_*.* \
  /Tivoli/bin/w32-ix86/LCF/test
```

2. Invoke the Configuration Script

```
/Tivoli/src/export/cfg/w32-ix86/Tivoli/downcall.cfg \
  test /Tivoli/bin w32-ix86 LCF
```

Note

.cfg script can be invoked once. If you want to re-execute .cfg script, you should:

1. Unregister the `Downcall` class.

```
wregister -u -r Classes Downcall
```

2. Modify the .cfg file. Find the `create_class_cli` string and change it to the `update_class_cli` string.

8.8.1.8 Create the Dependency

To create the dependency set, we performed the following steps.

1. Execute the dep script to configure the dependency.

```
cd /Tivoli/src/downsamp/platform/src
dep
```

For more information about the dep script, please refer to “dep” on page 340.

8.8.1.9 Install the Sample Application

Copy all files in the bin directory to the repository from the export directory as follows.

```
cp -r /Tivoli/src/export/bin /Tivoli
```

8.8.1.10 Run the Sample Application

To run the sample application, you can enter the following CLI command from the server or gateway, and the returned result is shown below.

```
dsmain <ep_name> <any_string>
```

```
bash$ dsmain hiro "This is sample string"
Making downcall with This is sample string
[THIS IS SAMPLE STRING]Making downcall for invoking upcall
bash$
```

8.8.2 View the Log Information

After running dsmain.exe, you can view the logged information. In this part of book, we introduce the logged messages in the log files and also explain the meaning of the messages.

8.8.2.1 Messages in the lcfd.log File During Downcall

The following messages appear in the lcfd.log file on the Endpoint when we invoke the sample application.

- When the `dsmain` starts, it calls remote method `method`. The new connection from the gateway is established.

```
Dec 17 09:34:27 Q lcfd New connection from 9.3.1.199+1148
```

- The `lcfd` daemon receives downcall request and detects that downcall server module does not exist on its cache.

```
Dec 17 09:34:27 Q lcfd Entering net_recv, receive a message
Dec 17 09:34:27 Q lcfd Leaving net_recv: bytes=203, (type=0 session=173464805)
Dec 17 09:34:27 2 lcfd CacheCheckExistence: NotFound:
C:\Tivoli\lcf\dat\1\cache\bin\w32-ix86\LCF\test\Downcall.exe
```

- The `lcfd` responds to the gateway.

```
Dec 17 09:34:27 Q lcfd Entering send_methstat
Dec 17 09:34:27 Q lcfd Entering send_struct
Dec 17 09:34:27 Q lcfd net_send of 52 bytes, session 173464805
Dec 17 09:34:27 Q lcfd Leaving send_struct
Dec 17 09:34:27 Q lcfd Leaving send_methstat
```

- The gateway downloads the server module, `Downcall.exe`.

```
Dec 17 09:34:27 Q lcfd Entering net_recv, receive a message
Dec 17 09:34:27 Q lcfd Leaving net_recv: bytes=85, (type=7 session=173464805)
Dec 17 09:34:27 2 lcfd reading:
C:\Tivoli\lcf\dat\1\cache\bin\w32-ix86\LCF\test\Downcall.exe
Dec 17 09:34:27 Q lcfd Entering net_recv, receive a message
Dec 17 09:34:27 Q lcfd Leaving net_recv: bytes=4618, (type=11 session=173464805)
```

* The `lcfd` daemon spawns new process and executes the downcall server.

```
Dec 17 09:34:27 Q lcfd setting-up inherit fd. netfd=240
Dec 17 09:34:27 1 lcfd Spawning:
C:\Tivoli\lcf\dat\1\cache\bin\w32-ix86\LCF\test\Downcall.exe, ses: 0a56dce5
```

* The `lcfd` daemon entering the listening state.

```
Dec 17 09:34:27 Q lcfd Entering Listener (running).
Dec 17 09:34:27 Q lcfd Entering net_wait_for_connection, timeout=-1 handle=0x3565d8
```

* According to the gateway's request, the `lcfd` daemon invokes the `method` method.

```
Dec 17 09:34:27 Q MethInit Entering mrt_run
Dec 17 09:34:27 Q MethInit argv: session_id=0a56dce5
Dec 17 09:34:27 Q MethInit Communication timeout set: 120.
Dec 17 09:34:27 Q MethInit Entering comm_reconnect
Dec 17 09:34:27 Q MethInit inherited fd. return from net_associated_fd. ipc=8533344,
netfd=240
Dec 17 09:34:27 Q MethInit Entering run_impl
Dec 17 09:34:27 Q MethInit Entering send_methstat
Dec 17 09:34:27 Q MethInit Entering send_struct
Dec 17 09:34:27 Q MethInit net_send of 52 bytes, session 173464805
Dec 17 09:34:27 Q MethInit Leaving send_struct
Dec 17 09:34:27 Q MethInit Leaving send_methstat
Dec 17 09:34:27 Q MethInit waiting for input args
Dec 17 09:34:27 Q MethInit Entering net_recv, receive a message
Dec 17 09:34:27 Q MethInit Leaving net_recv: bytes=273, (type=3 session=173464805)
Dec 17 09:34:27 2 MethInit Looking for method: method.
```

* The `method` method is called.

```
Dec 17 09:34:27 Q method calling method.
```

* The following is a only log message that is queued by `method`.

```
Dec 17 09:34:27 Q method Entering downcall method
```

* Response to the gateway.

```
Dec 17 09:34:27 Q method method returned.
Dec 17 09:34:27 Q method send_results (max/len) 80/38
Dec 17 09:34:27 Q method Entering send_methstat
Dec 17 09:34:27 Q method Entering send_struct
Dec 17 09:34:27 Q method net_send of 92 bytes, session 173464805
Dec 17 09:34:27 Q method Leaving send_struct
Dec 17 09:34:27 Q method Leaving send_methstat
```

* Shutdown server. Note that `method` is "per-method" server type.

```
Dec 17 09:34:27 2 method Clean Shutdown method.
```

- When dsmain calls next remote method invoke, new connection is established from the gateway.

```
Dec 17 09:34:27 Q lcfd New connection from 9.3.1.199+1149
```

- The lcfd daemon checks dependencies related to invoke method. The invoke method itself is in Downcall.exe. Additionally, it checks existence of upcall.exe that is specified as a dependency of Downcall::invoke as discussed in previous section.

```
Dec 17 09:34:27 Q lcfd Entering net_recv, receive a message
Dec 17 09:34:27 Q lcfd Leaving net_recv: bytes=282, (type=0 session=173464806)
Dec 17 09:34:27 2 lcfd CacheCheckExistence: Found: C:\Tivoli\lcf\dat\1\cache\bin\w32-ix86\LCF\test\
Dec 17 09:34:27 2 lcfd CacheCheckExistence: NotFound: C:\Tivoli\lcf\dat\1\cache\bin\w32-ix86\LCF\tes
Dec 17 09:34:27 Q lcfd Entering send_methstat
Dec 17 09:34:27 Q lcfd Entering send_struct
Dec 17 09:34:27 Q lcfd net_send of 52 bytes, session 173464806
Dec 17 09:34:27 Q lcfd Leaving send_struct
Dec 17 09:34:27 Q lcfd Leaving send_methstat
```

- The gateway downloads the dependency, upcall.exe.

```
Dec 17 09:34:27 Q lcfd Entering net_recv, receive a message
Dec 17 09:34:27 Q lcfd Leaving net_recv: bytes=83, (type=7 session=173464806)
Dec 17 09:34:27 2 lcfd reading: C:\Tivoli\lcf\dat\1\cache\bin\w32-ix86\LCF\test\upcall.exe
Dec 17 09:34:27 Q lcfd Entering net_recv, receive a message
Dec 17 09:34:27 Q lcfd Leaving net_recv: bytes=4106, (type=11 session=173464806)
```

- The lcfd daemon spawns new process and executes upcall client.

```
Dec 17 09:34:27 Q lcfd setting-up inherit fd. netfd=192
Dec 17 09:34:27 1 lcfd Spawning: C:\Tivoli\lcf\dat\1\cache\bin\w32-ix86\LCF\test\Downcall.exe, ses:
Dec 17 09:34:27 Q lcfd Entering Listener (running).
Dec 17 09:34:27 Q lcfd Entering net_wait_for_connection, timeout=-1 handle=0x3565d8
```

- According to gateway's request, the lcfd daemon invokes invoke method.

```
Dec 17 09:34:27 Q MethInit Entering mrt_run
Dec 17 09:34:27 Q MethInit argv: session_id=0a56dce6
Dec 17 09:34:27 Q MethInit Communication timeout set: 120.
Dec 17 09:34:27 Q MethInit Entering comm_reconnect
Dec 17 09:34:27 Q MethInit inherited fd. return from net_associated_fd. ipc=8533344, netfd=192
Dec 17 09:34:27 Q MethInit Entering run_impl
Dec 17 09:34:27 Q MethInit Entering send_methstat
Dec 17 09:34:27 Q MethInit Entering send_struct
Dec 17 09:34:27 Q MethInit net_send of 52 bytes, session 173464806
Dec 17 09:34:27 Q MethInit Leaving send_struct
Dec 17 09:34:27 Q MethInit Leaving send_methstat
Dec 17 09:34:27 Q MethInit waiting for input args
Dec 17 09:34:27 Q MethInit Entering net_recv, receive a message
Dec 17 09:34:27 Q MethInit Leaving net_recv: bytes=273, (type=3 session=173464806)
Dec 17 09:34:27 2 MethInit Looking for method: invoke.
```

• The invoke method is called.

```
Dec 17 09:34:27 Q invoke calling method.
```

• The following line is logged by invoke method.

```
Dec 17 09:34:27 Q invoke Entering downcall method for invoking upcall
```

• At this timing, invoke method spawns new process and execute upcall.exe program.

```
Dec 17 09:34:27 Q invoke Exit downcall method for invoking upcall
```

• Respond to the gateway.

```
Dec 17 09:34:27 Q invoke method returned.
Dec 17 09:34:27 Q invoke send_results (max/len) 80/6
Dec 17 09:34:27 Q invoke Entering send_methstat
Dec 17 09:34:27 Q invoke Entering send_struct
Dec 17 09:34:27 Q invoke net_send of 60 bytes, session 173464806
Dec 17 09:34:27 Q invoke Leaving send_struct
Dec 17 09:34:27 Q invoke Leaving send_methstat
```

• Shutdown the "per-method" server invoke.

```
Dec 17 09:34:27 2 invoke Clean Shutdown invoke.
```

• The Upcall.exe connects lcfd daemon.

```
Dec 17 09:34:27 Q lcfd New connection from 127.0.0.1+1151
```

- The `lcfd` daemon receives upcall request from upcall.exe.

```
Dec 17 09:34:27 Q lcfd Entering net_recv, receive a message
Dec 17 09:34:27 Q lcfd Leaving net_recv: bytes=173, (type=16 session=0)
Dec 17 09:34:27 Q lcfd UPCALL_START request
```

- The `lcfd` daemon sends the upcall request to the gateway.

```
Dec 17 09:34:27 2 lcfd Connecting to '9.3.1.199+9494'
Dec 17 09:34:27 Q lcfd net_send of 426 bytes, session 0
Dec 17 09:34:27 Q lcfd Entering send_struct
Dec 17 09:34:27 Q lcfd net_send of 207 bytes, session 116
Dec 17 09:34:27 Q lcfd Leaving send_struct
```

- The `lcfd` daemon waits for next request form anywhere.

```
Dec 17 09:34:27 Q lcfd Entering Listener (running).
Dec 17 09:34:27 Q lcfd Entering net_wait_for_connection, timeout=-1 handle=0x3565d8
```

8.8.2.2 Messages in the lcfd.log File During Upcall

The following messages also appear in the lcfd.log file on the Endpoint, and these messages are logged by the upcall.exe.

- The upcall.exe is invoked on the Endpoint.

```
Dec 17 09:34:27 1 upcall Starting Upcall
```

- The Upcall.exe calls the remote method `method`.

```
Dec 17 09:34:27 Q upcall Entering t_generic_stub in ecp/upcall.c
Dec 17 09:34:27 Q upcall LCF upcall capability initialized
Dec 17 09:34:27 Q upcall Warning: no upcall user or credential specified
```

- To send the upcall request to the gateway, the stub contacts to the `lcfd` daemon.

```
Dec 17 09:34:27 Q upcall contacting lcfd with upcall method
Dec 17 09:34:27 2 upcall Connecting to '127.0.0.1+9493'
Dec 17 09:34:27 Q upcall Entering send_struct
Dec 17 09:34:27 Q upcall net_send of 173 bytes, session 0
Dec 17 09:34:27 Q upcall Leaving send_struct
Dec 17 09:34:27 Q upcall Entering net_recv, receive a message
```

- The gateway connects to upcall.exe.

```
Dec 17 09:34:27 Q upcall Leaving net_recv: bytes=426, (type=13 session=0)
Dec 17 09:34:27 Q upcall Entering net_wait_for_connection, timeout=0 handle=0x3528a0
Dec 17 09:34:27 Q upcall New connection from 9.3.1.199+1153
```

- The gateway returns a message.

```
Dec 17 09:34:27 Q upcall waiting to send upcall params
Dec 17 09:34:27 Q upcall Entering net_recv, receive a message
Dec 17 09:34:27 Q upcall Leaving net_recv: bytes=28, (type=17 session=173464807)
Dec 17 09:34:27 Q upcall net_send of 48 bytes, session 173464807
Dec 17 09:34:27 Q upcall waiting for upcall results
Dec 17 09:34:27 Q upcall Entering net_recv, receive a message
Dec 17 09:34:28 Q upcall Leaving net_recv: bytes=114, (type=5 session=173464807)
```

- The Upcall.exe writes log records and terminates.

```
Dec 17 09:34:28 1 upcall out_string="Message from gateway[This is sample string]"
Dec 17 09:34:28 1 upcall Ending Upcall
```

8.8.2.3 Messages in the gatelog File

The following messages appear in the gatelog file on the Endpoint gateway when we invoke the sample application.

- When dsmain starts, the gateway receives a request.

```
1998/12/17 09:34:27 +06: sched: got a job
```

- The gateway checks the dependency.

```
1998/12/17 09:34:27 +06: gwcache: hit key=<1157485836.1.1987,.meth.,method>
1998/12/17 09:34:27 +06: downcall: Method body /bin/w32-ix86/LCF/test/Downcall found.
```

- The gateway sends the downcall request to the Endpoint.

```
1998/12/17 09:34:27 +06: new_session: a56dce5, connecting to 9.3.1.199+9493...
1998/12/17 09:34:27 +06: reader_thread: received data: session=a56dce5, type=5, len=92
1998/12/17 09:34:27 +06: destroying session a56dce5
```

- The gateway receives the next request `invoke` from `dsmain`.

```
1998/12/17 09:34:27 +06: sched: got a job
```

- The gateway checks the dependencies.

```
1998/12/17 09:34:27 +06: gwcache: hit key=<1157485836.1.1987,.meth.,invoke>
1998/12/17 09:34:27 +06: gwcache: hit key=<1157485836.1.2183#Depends::Mgr#,.attr.,_info>
1998/12/17 09:34:27 +06: downcall: Method body /bin/w32-ix86/LCF/test/Downcall found.
1998/12/17 09:34:27 +06: downcall: dependency /bin/w32-ix86/LCF/test/upcall found.

1998/12/17 09:34:27 +06: new_session: a56dce6, connecting to 9.3.1.199+9493...
1998/12/17 09:34:27 +06: reader_thread: received data: session=a56dce6, type=5, len=60
1998/12/17 09:34:27 +06: destroying session a56dce6
```

- The gateway receives the upcall request from the Endpoint.

```
1998/12/17 09:34:27 +06: reconnect_thread: connection from 9.3.1.199+1152
1998/12/17 09:34:27 +06: tcp server: waiting for connection on 0.0.0.0+9494...
1998/12/17 09:34:27 +06: reader_thread: received data: session=74, type=16, len=207
```

- The `Upcall::method` is called.

```
1998/12/17 09:34:27 +06: sched: got a job
1998/12/17 09:34:27 +06: new_session: a56dce7, connecting to 9.3.1.199+1150...
1998/12/17 09:34:27 +06: upcall start: from=9.3.1.199+1150, class=Upcall, method=method
1998/12/17 09:34:27 +06: gwcache: miss key=<1157485836.1.1095,.meth.,method>
1998/12/17 09:34:27 +06: gwcache: hit key=<1157485836.1.1095,.inh.,>
1998/12/17 09:34:27 +06: gwcache: miss key=<1157485836.1.1097,.meth.,method>
1998/12/17 09:34:27 +06: gwcache: hit key=<1157485836.1.1097,.inh.,>
1998/12/17 09:34:27 +06: gwcache: hit key=<1157485836.1.1096,.meth.,method>
1998/12/17 09:34:27 +06: gwcache: miss key=<1157485836.1.1095,.groups.,>
1998/12/17 09:34:28 +06: destroying session a56dce7
```

Chapter 9. Management Examples Using TMA

In this chapter, we introduce some management examples with the TMA and also discuss solutions for large environments and the tools that we developed during the project. These tools should be useful for many customers.

9.1 Managing Enterprise Environment with TMA

We discussed the considerations and advantages of the principal Tivoli Management Applications with the TMA. However, in a customer's environment, we have to understand not only the features of the Tivoli Management Applications, but also the whole design, including the TMR, the network, and so on. This section provides ideas and know-how for managing a large-scale environment remotely. The examples used in this explanation should be useful when you plan, design, or implement the TMA in a large-scale environment using Tivoli Management Applications.

9.1.1 Allocation of EP Manager, EP Gateway and EP

For managing very large distributed system environments, we need to carefully consider the TMR design. Version 3.6 of the Tivoli Management Framework provides a three-tiered management structure and it has made stable and extensible managing possible. If we can allocate the management resources to the proper locations, Version 3.6 of the Tivoli Management Framework also extends the reliability and availability of the management system.

In this section, we introduce some examples for understanding the most efficient allocation of the Endpoint Manager, the Endpoint gateway and the Endpoint. Reading this section should help you make sense of how to allocate the Endpoint Manager, Endpoint gateway and Endpoint for managing an enterprise-scale environment.

Note

This section assumes you want to manage a very large environment remotely using the Tivoli Management Applications with the TMA.

9.1.1.1 Example 1: Single TMR Configuration

Basically, a single Endpoint gateway can handle up to around 2000 Tivoli Management Agents (Endpoints) depending on the hardware configuration. This means a single TMR can manage several thousand Tivoli Management

Agents. It is absolutely different from the prior TMR structure because a single TMR server could manage only 200 Managed Nodes. This example is shown in Figure 119, which shows the most simple but efficient configuration.

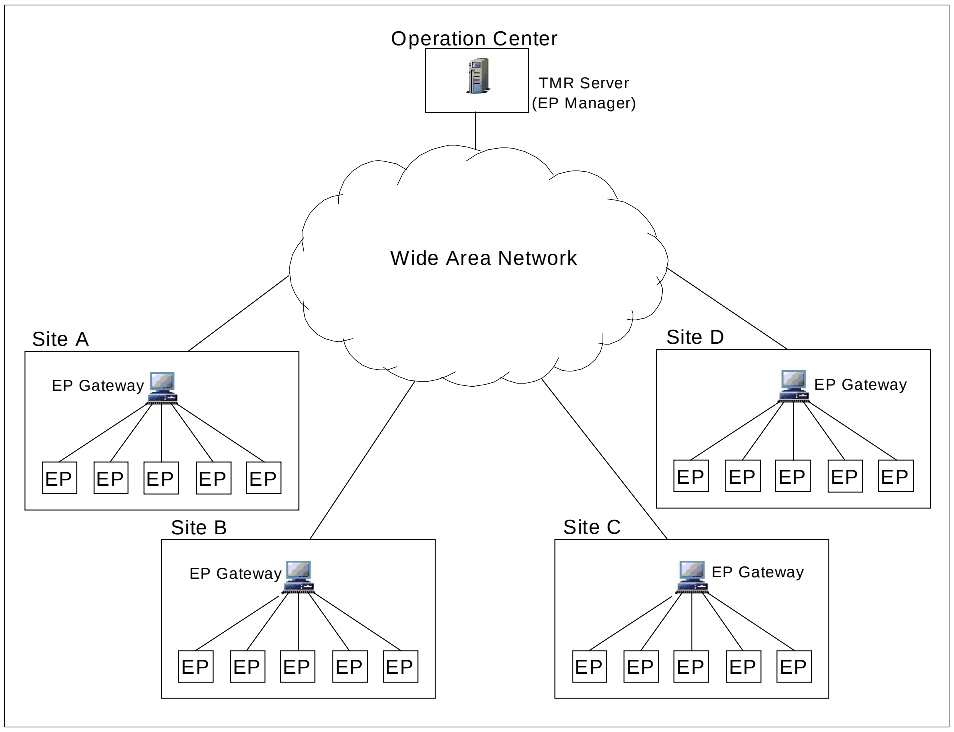


Figure 119. The Single TMR Configuration

In this example, there are a few important considerations, as follows.

Allocate the EP Gateway and the EPs to the same location:
As we mentioned before, in the LCF architecture, the Endpoint methods are optimized to require less communication with the TMR server because the Endpoint gateway performs authentication, location and inheritance (ALI) functions for the Endpoints. This means if the Endpoint has stored a method in its cache, the Endpoint can invoke the method without a call to the TMR server (Endpoint Manager).

For example, Distributed Monitoring runs the Sentry engine as a single method (process) on the each TMA. This engine monitors and logs the system conditions on the TMA if you create and distribute the Sentry monitor to the TMA. In this case, if the network between the Endpoint Manager and Endpoint gateway goes down, the Sentry monitor, which is running on the

Endpoint, should be able to keep monitoring. As you know, the Sentry monitor cannot run when the network between a TMR server and the Managed Node goes down, even if the Managed Node is available. The Distributed Monitoring case is a typical example.

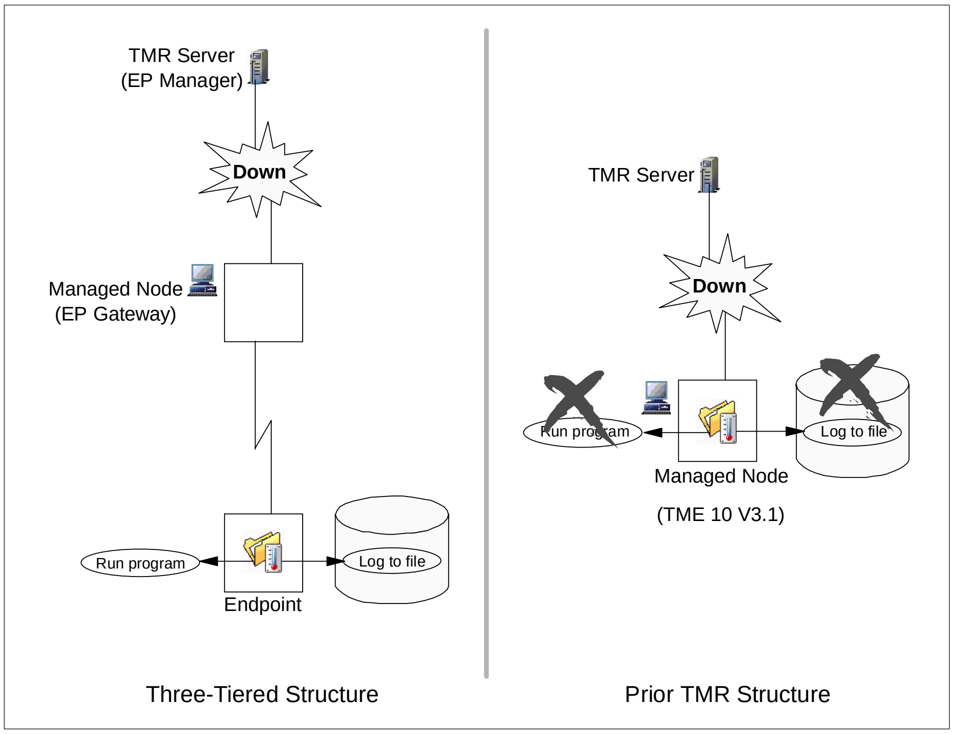


Figure 120. The Advantage of Three-Tiered Structure

Therefore, if possible, the Endpoint gateway and Endpoint should be located in the same location (such as the same building, branch) connected via LAN because the LAN is more reliable than the WAN. It may depend on the customer's system or business environment, but it improves the availability of the management system. We will talk about this kind of situation in detail later.

The most important point here is the location of the Endpoint gateway and the Endpoints. To improve the performance and reliability of the management system, we have to consider the appropriate allocation of the Endpoint

Manager, Endpoint gateways and Endpoints for each customer. The following table shows a comparison of networks.

Table 59. Comparison Between LAN and WAN

	Local Area Network (LAN)	Wide Area Network (WAN)
Speed	Fast	Slow
Throughput	High	Low
Reliability	High	Low

After understanding the characteristics of the networks, we have to consider the following issues as well:

- Using the MDist fan out function effectively
- Improving throughput between the Endpoint gateway and Endpoints
- Avoiding Endpoint isolation

As you can see from the table, the LAN is more reliable than the WAN and also faster than the WAN. By default, the Endpoint gateways are configured as an MDist repeater automatically. This means if we use a slow network like a WAN between the Endpoint Manager and Endpoint gateway, it would not cause a serious problem because we can use the MDist fan out function. To use the MDist fan out function efficiently, the network between the Endpoint Manager and Endpoint gateway could be configured with a slow link (such as WAN) and the network between the Endpoint gateway and Endpoint should be configured with a fast link (such as LAN).

If the network between the Endpoint gateway and Endpoint becomes unavailable, what happens? As you know, the Endpoint is isolated and attempts to find other available Endpoint gateways. This could potentially make management operations confusing. To avoid it, the Endpoint gateway and Endpoints should be located in the same place and connected to each other through a LAN. It obviously improves the throughput between the Endpoint gateway and Endpoints as well.

The Number of Endpoint Gateways in a TMR:
The number of available Endpoint gateways for one Endpoint is a very important issue in managing very large environments because the Endpoint is really configurable for Endpoint login. This means that the Endpoint can log into other available Endpoint gateways that are defined in the Endpoint login interfaces list, even if the assigned gateway is unreachable. However, if you do that, you need to configure the Endpoint login interfaces list when you

install the Endpoint; so you should understand the real environment and what happens when the Endpoint gateway goes down.

Note

In general, don’t use the broadcast for the Endpoint login. Especially in a large environment, it could cause a serious problem to other applications or systems. Therefore, you should configure the broadcast as disabled (`bcase_disable=1`) in your environment.

From this point of view, the number of available Endpoint gateways should be at least three in a single TMR. Of course, many available Endpoint gateways are much better than a few available Endpoint gateways.

Creating an Endpoint Gateway on the TMR Server:
In some cases, this might be recommended for the single TMR configuration, especially for smaller TMRs. By adding an Endpoint gateway to the TMR server, this provides another secondary gateway that could be used if the primary gateways fail. For example, in the following situation where all Endpoint gateways become unavailable (shown in Figure 121 on page 364), the Endpoint Manager can be the Endpoint gateway if you created it on the Endpoint Manager. Normally, the Endpoint Manager (TMR server) is configured on the most reliable machine, so that this configuration makes the single TMR environment more reliable. For performance reasons, you may not want the gateway on the TMR Server to be used in normal operation.

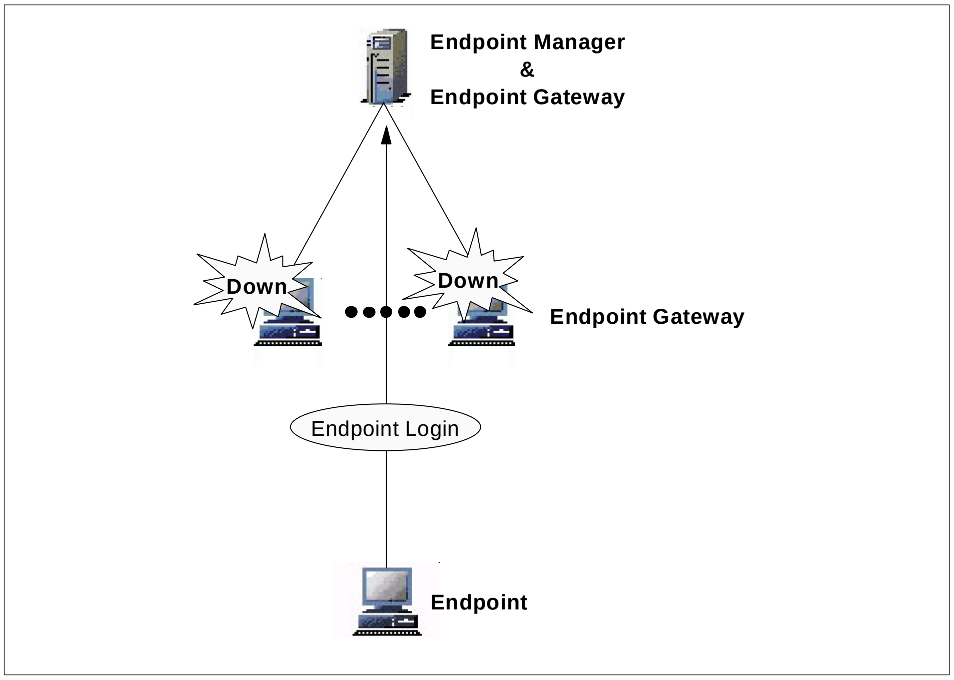


Figure 121. Creating the Endpoint Gateway on the Endpoint Manager

9.1.1.2 Example 2: Multiple Endpoint Gateway Configuration

This configuration is one of the most recommended ways to manage a large environment. There is more than one Endpoint gateway at each site; so each Endpoint can log into more than one Endpoint gateway through the Local Area Network (LAN). It provides a higher level of availability when an assigned gateway becomes unavailable, or when you need to shutdown an Endpoint gateway machine for some reason (such as for machine maintenance).

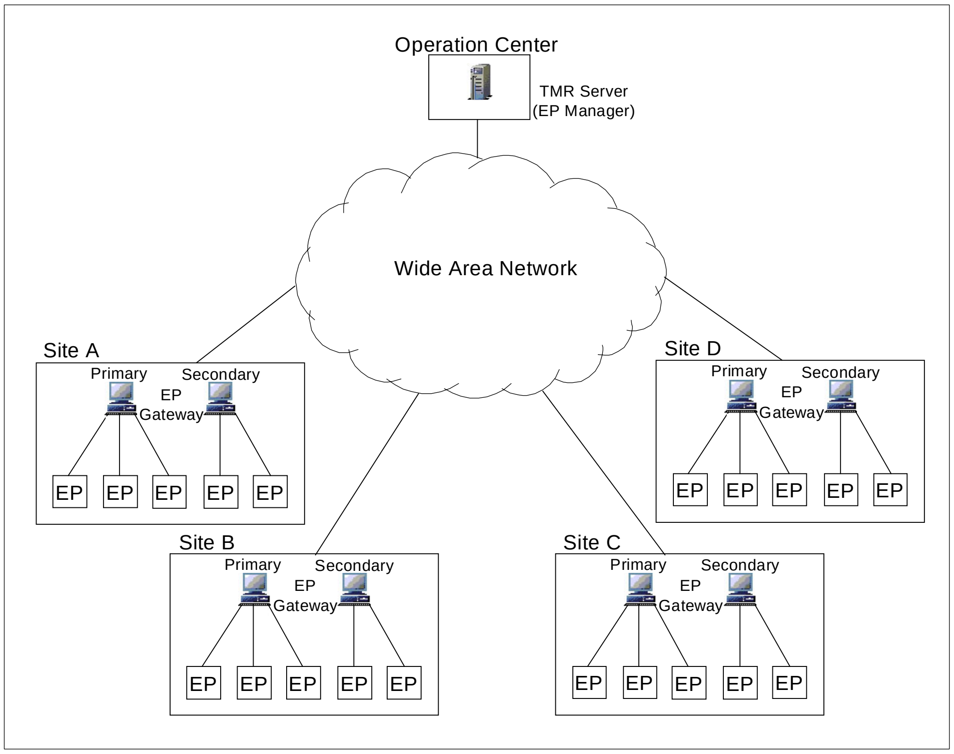


Figure 122. Multiple Endpoint Gateways Configuration

In the TMA environment, the Endpoint gateway plays a very important role as the mid-level manager. Of course, the Endpoint Manager (TMR server) is still the most important component in the Tivoli management environment. However, the Endpoint Manager does not allow us to configure an alternate Endpoint Manager. In other words, duplicate configuration is not supported for the Endpoint Manager (TMR server). (Unless of course you use a third party High Availability project.) But the Endpoint gateway allows us to configure alternate gateways. To keep high availability, you may want to consider creating more than one Endpoint gateway at each site.

If your environment or customer environment does not allow you to configure two Endpoint gateways at each site, the following can be a solution for this situation:

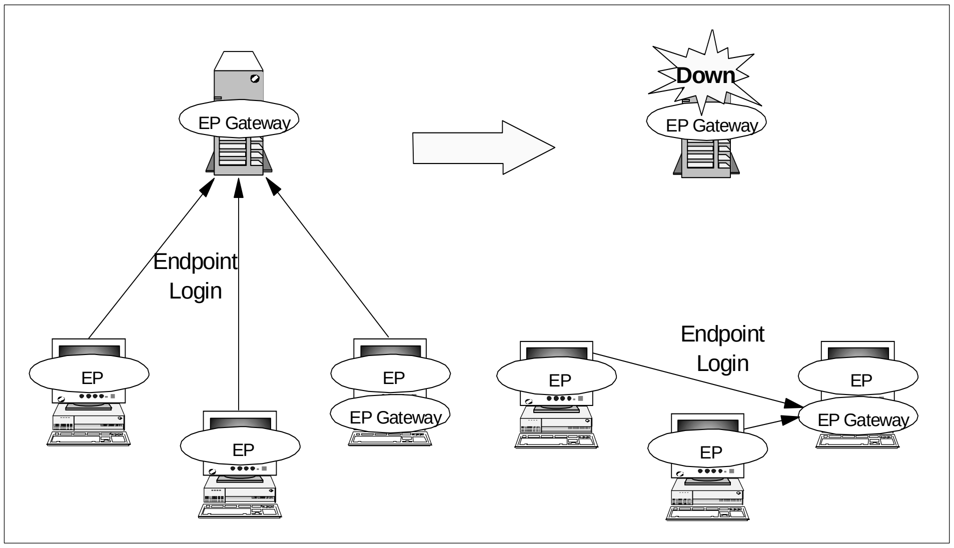


Figure 123. Creating the Endpoint Gateway on the Endpoint

As you can see, in this case, the Endpoint gateway is created on the Endpoint. The Endpoint gateway, which is created on the Endpoint, will be used only for an emergency because the load of the Endpoint gateway process is not low. Recall that Endpoint gateways must reside on Managed Nodes. So the Endpoint you choose, must also be installed as a Managed Node. Though since you won't use the Managed Node interface for managing the system, the overhead involved will not be very much.Therefore, when you define this configuration, you should take care to select the machine on which the Endpoint gateway is created.

In this configuration, you have to take care of the port number which the Endpoint and Endpoint gateway used on a single machine as well. By default, the Endpoint and Endpoint gateway attempt to use the same port number, 9494. If you don't correct this, a port busy error will occur when the Endpoint or Endpoint gateway attempts to open the port and the process which opens the port successfully first becomes available. Of course, the process which cannot open the port is not available. This just depends on the timing. For no logical reason, the Endpoint process usually opens the port 9494 successfully first in our environment.

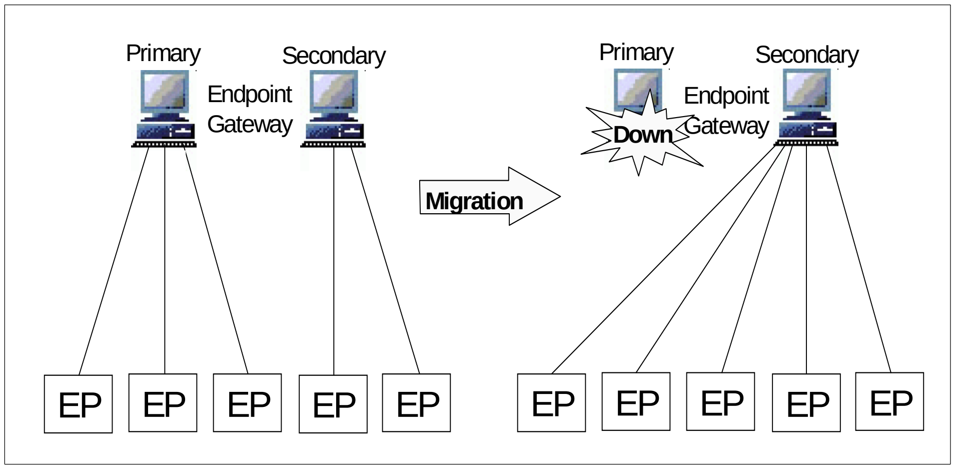


Figure 124. Endpoint Gateway Migration Operation

In the multiple Endpoint gateways configuration, we recommend that you configure the primary Endpoint gateway and the secondary Endpoint gateway at each site as shown in Figure 124. Usually, when both Endpoint gateways are working fine, half of the Endpoints at the site are managed by the primary Endpoint gateway and the rest of the Endpoints are managed by the secondary Endpoint gateway. However, when the primary Endpoint gateway is needed to be shutdown for maintenance or the primary Endpoint gateway goes down, the secondary Endpoint gateway manages all Endpoints in the site.

In a real environment, we sometimes need to shutdown the Endpoint gateway machine for machine maintenance. For this operation, we developed a tool that migrates the Endpoints that have logged into the specified Endpoint gateway to the another Endpoint gateway specified as the argument. We will introduce this tool in the section “Useful Tools for Using TMA” on page 377.

9.1.1.3 Example 3: Multiple TMR Configuration

A multiple TMR configuration is another solution for managing a large environment. In the two-tiered management structure environment, it was the only solution for managing a large environment, because a single TMR server can manage only about 200 Managed Nodes. However, in the three-tiered management structure, as we mentioned, there are multiple solutions for managing the large environment, and the multiple TMR configuration is one of these solutions. To implement the multiple TMR configuration, we have to connect two TMRs. However, this increases the complexity of your environment. For this reason we don’t strongly recommend that you implement the multiple TMR configuration. Particularly in a large environment, the information and data that are managed and stored in the Tivoli object databases are very large. In such a situation, it is more difficult to keep the consistency of the databases in the multiple TMR environment.

The following figure shows an example of a multiple TMR configuration. In this case, each TMR server is configured as equal. In other words, each TMR server is managing almost the same number of Endpoint gateways and Endpoints. This configuration is a classic configuration for multiple TMR environments. This configuration might be implemented in the following situations:

- There are two geographically dispersed locations with somewhat autonomous management organizations.
- The customer has already implemented a multiple TMR configuration using the prior version of Tivoli and migrates to Version 3.6 of Tivoli using TMA.
- To off-load TMR servers in large environments.

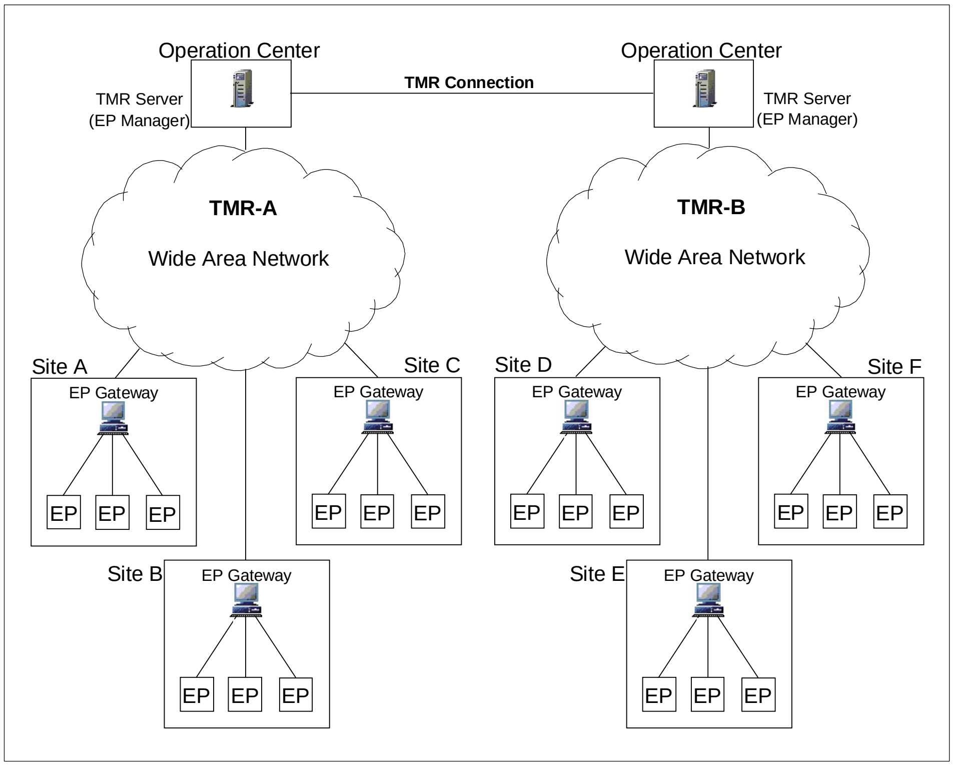


Figure 125. Multiple TMR Configuration

The following figure shows another example of the multiple TMR configuration. In this case, one TMR server is configured to contain the managed resources and the other controls the management applications. This type of configuration is often referred to as a hub and spoke, since the applications are maintained on the hub TMR and managed resources could be members of one or more spoke TMRs.

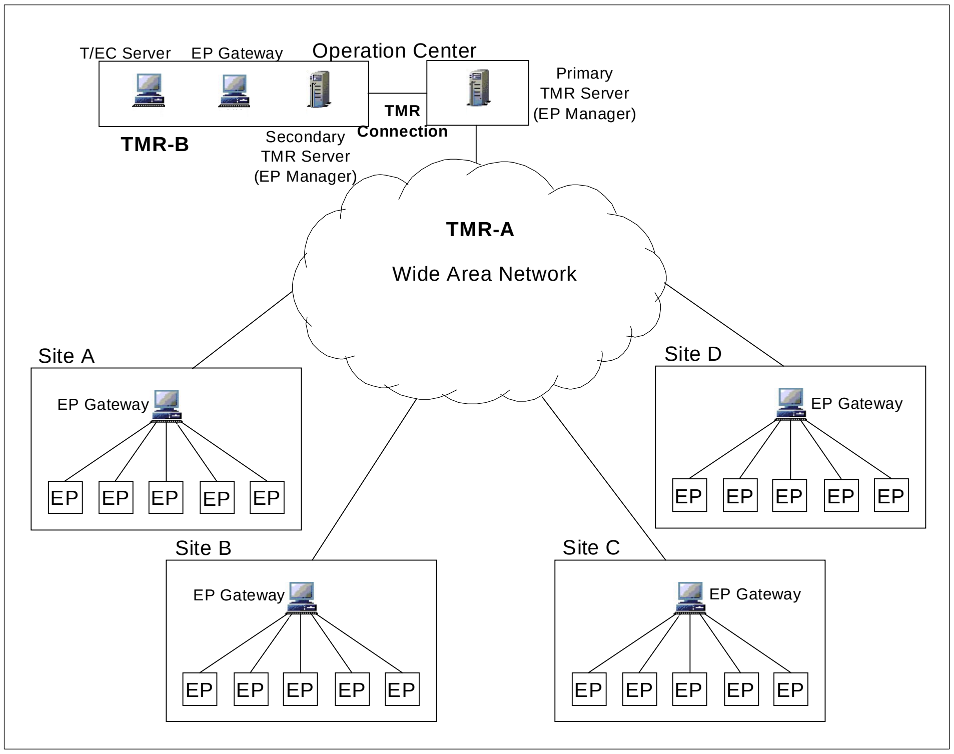


Figure 126. Multiple TMR Configuration (Primary TMR and Secondary TMR)

In this case, when the primary TMR server (Endpoint Manager) becomes unavailable, the secondary TMR server (Endpoint Manager) can take over managing TMAs from the primary server if you define the appropriate Endpoint login interfaces list. This sample configuration gives you high availability for managing thousands of TMAs.

Note

To take over managing the TMAs from the primary server to the secondary server, as in this case, you need not only to define the appropriate Endpoint login interfaces list, but also disable the Endpoint gateway process in the primary TMR. Then the Endpoint will have an isolated status, so that it attempts to perform an initial login to the Endpoint gateways which are defined in its login interfaces list. If the Endpoint gateway in the secondary TMR is defined in the Endpoint login interfaces list, the Endpoint logs into the Endpoint gateway in the secondary TMR. After the takeover, the Endpoints can be managed by the secondary TMR server using Tivoli Management Applications. Of course, the profiles used by the applications should be defined in the secondary TMR as well.

9.1.2 High Availability Solution for TMR Server

As you know, unfortunately, the TMR server can be a single point of failure even with Version 3.6 of Tivoli. In this chapter, we introduced some examples to improve the availability for managing TMAs.

To duplicate the TMR server function completely, you need to use software which provides high availability functions. IBM HACMP for AIX is such a product. This kind of software normally needs special hardware to duplicate the hard disk or perform heartbeating, so that there are some considerations when you implement such a solution.

The High Availability software allows you to take over not only the hardware, but also the applications which run on the primary server. If you implement the high availability solution using software in the sample configuration which we introduced, the management system can be extremely reliable.

9.2 Endpoint Login Interfaces List Configuration

The Endpoint login interfaces list is the most important option for the Endpoint because it affects the Endpoint gateway selection. Another important configuration for the Endpoint gateway selection is the select_gateway_policy policy. This policy allows you to configure the assigned gateway automatically when the Endpoint Manager and Endpoint gateways are working correctly. However, if the Endpoint Manager in the primary TMR becomes unavailable, or all Endpoint gateways become unavailable in the primary TMR, the select_gateway_policy may not be used for the Endpoint gateway selection.

In this case, we assume that broadcast is configured as disabled. If we define only one Endpoint gateway in the Endpoint login interfaces list and then the Endpoint gateway becomes unavailable, what happens? The Endpoint cannot log into any Endpoint gateways unless the assigned gateway becomes available. Of course, the Endpoint cannot use the select_gateway_policy in this case because the Endpoint needs to use the broadcast for finding an intercepting gateway, but the broadcast is configured as disabled. From this point of view, you have to configure the appropriate Endpoint login interfaces list, especially in a large environment.

Example 1 Multiple Endpoint Gateways:
The following figure shows an example of one recommended TMR design for managing TMAs in a large environment. In this case, for backup purposes, we created the Endpoint gateway in the operation center where the Endpoint Manager is located. The TEC server is configured on a different machine from the Endpoint Manager for improved performance.

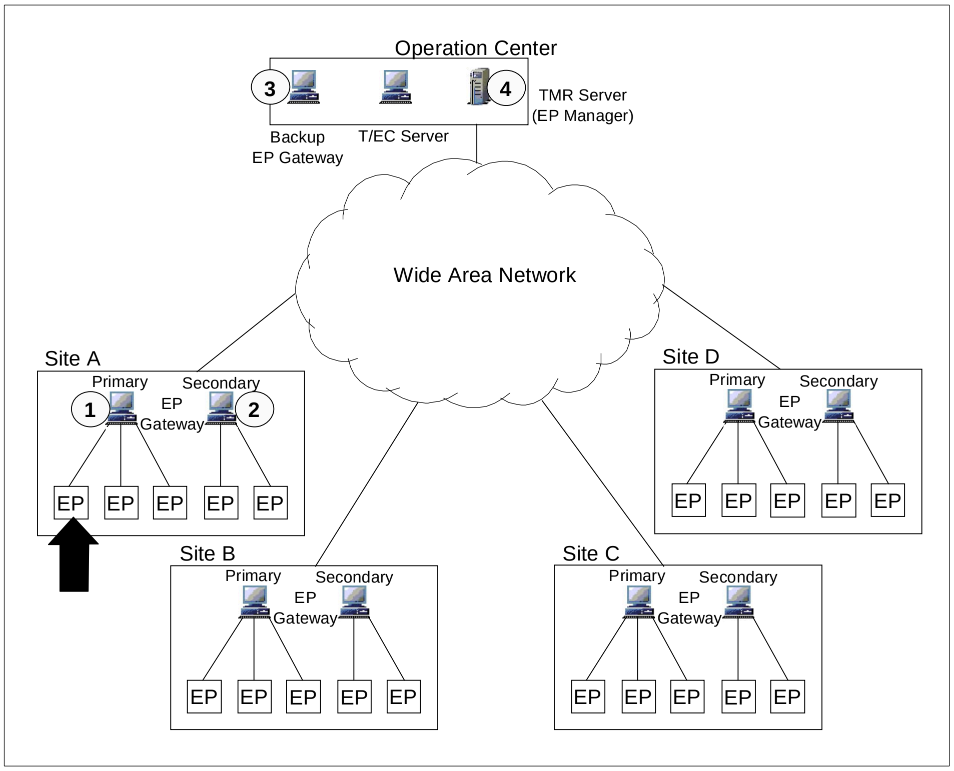


Figure 127. The Multiple Endpoint Gateways Configuration Sample

In this example (Figure 127 on page 372), we indicate the recommended Endpoint login interface list shown in the figure by the arrow. The following are the Endpoint gateways that should be configured as the alternate gateway in the login interfaces list:

1. The primary Endpoint gateway in the same site.
2. The secondary Endpoint gateway in the same site.
3. The backup Endpoint gateway in the operation center.
4. The Endpoint gateway that is created on the Endpoint Manager.

Example 2 Single TMR:
This example shows a simple TMR design. To improve availability, we recommend you create more Endpoint gateways in the TMR. In this case, the recommended login interfaces configuration is shown in Figure 128.

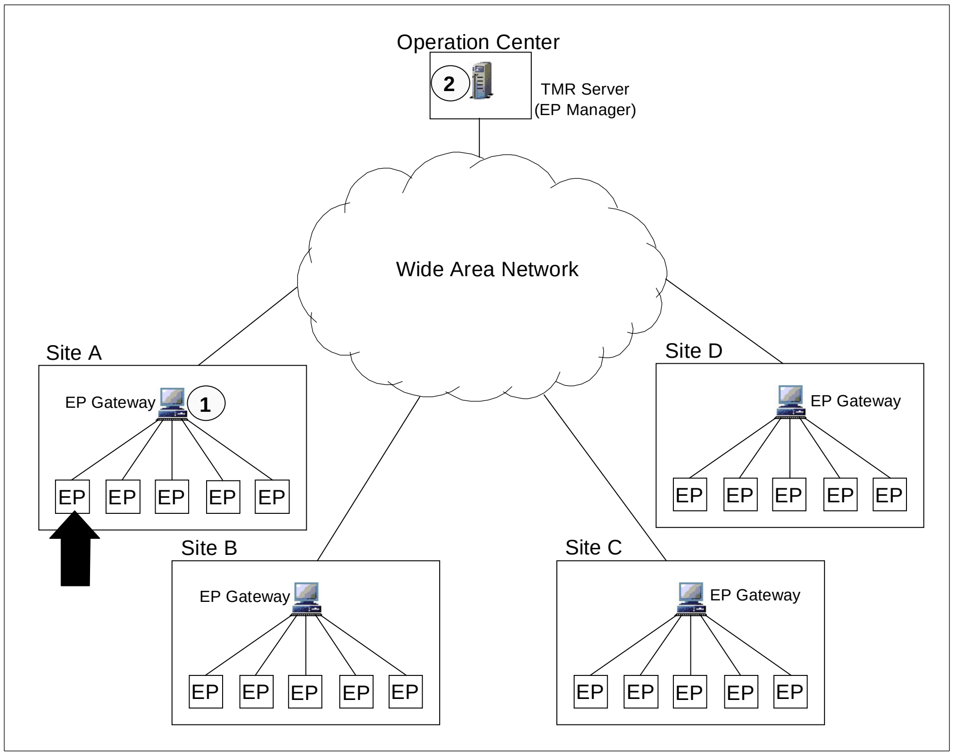


Figure 128. The Single TMR Configuration Sample

1. The Endpoint gateway at the same site.
2. The Endpoint gateway that is created on the Endpoint Manager.

Example 3 Multiple TMR Configuration:
This example shows the classic multiple TMR design. To improve availability, we recommend that you create an Endpoint gateway in each TMR. In this case, the recommended login interfaces configuration is shown in Figure 129.

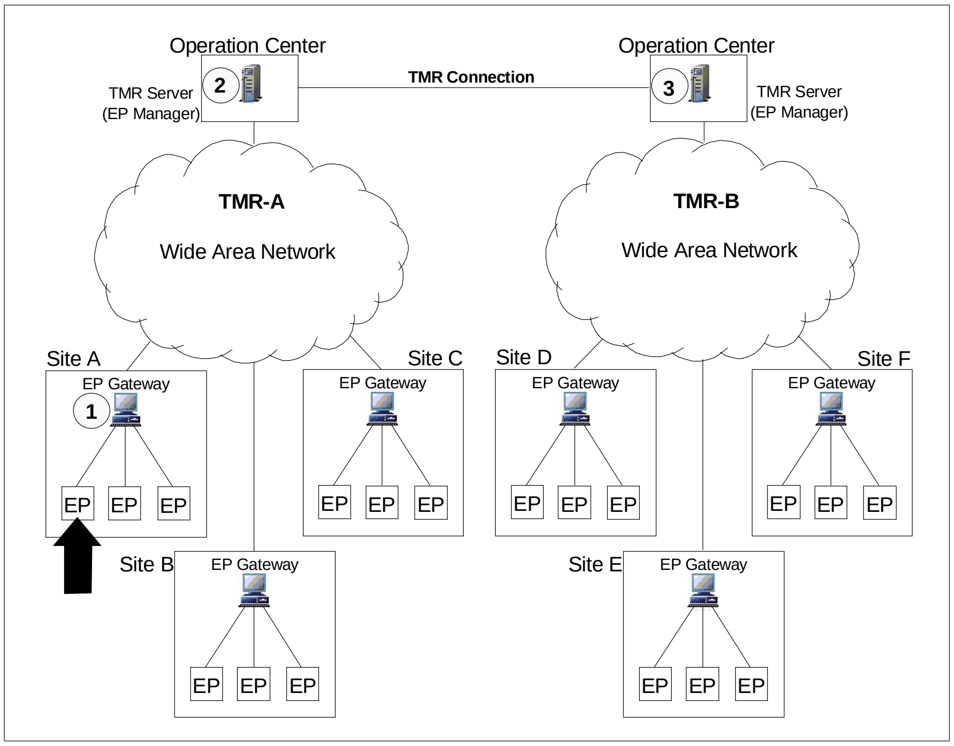


Figure 129. The Classic Multiple TMR Configuration Sample

1. The Endpoint gateway at the same site.
2. The Endpoint gateway that is created on the Endpoint Manager in the same TMR (TMR-A).
3. The Endpoint gateway that is created on the Endpoint Manager in another TMR (TMR-B).

It is important to note that in the case where an Endpoint logs in into a gateway in a different TMR, it will have a different OID and therefore the current profile manager subscriber lists will not work to manage the Endpoint. If you decide to implement this back up scenario, you will need to put Endpoint policy scripts in place that will automatically create new subscriber lists as Endpoints failover to the backup TMR. This could be complicated and should be implemented with care.

Example 4 Multiple TMR Configuration:
This example shows a multiple TMR design. To improve availability, we recommend that you create one more Endpoint gateway in each site. In this case, the recommendable login interface configuration is shown in Figure 130 on page 375.

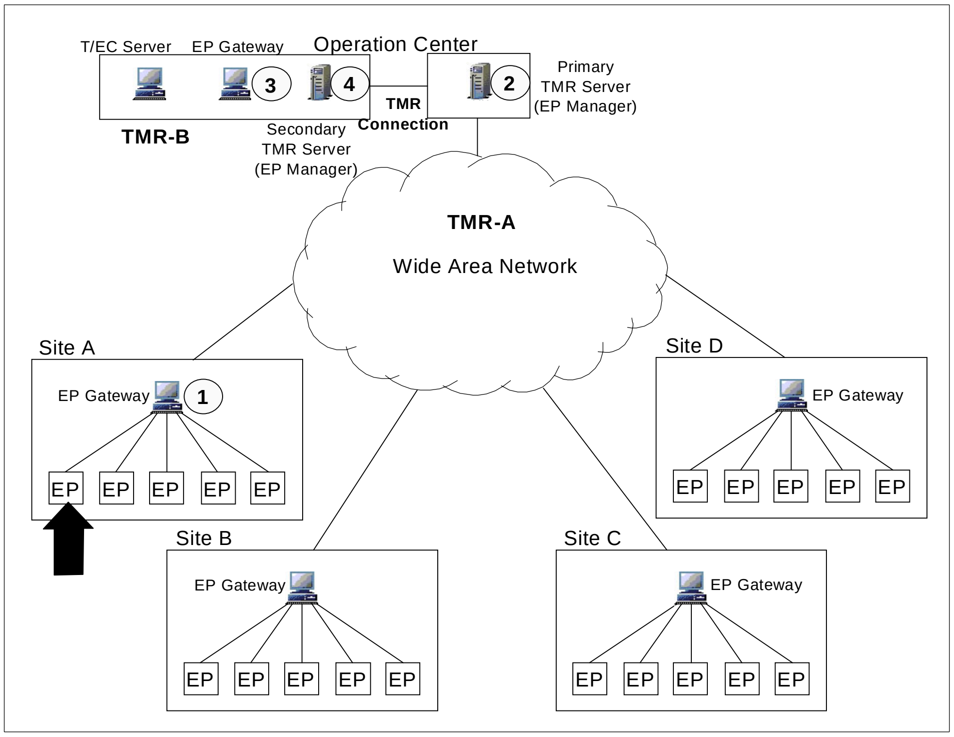


Figure 130. The Multiple TMR Configuration Sample

1. The Endpoint gateway at the same site.
2. The Endpoint gateway that is created on the Endpoint Manager in the same TMR (TMR-A).
3. The backup Endpoint gateway in another TMR (TMR-B).
4. The Endpoint gateway that is configured on the Endpoint Manager in another TMR (TMR-B).

Note

In each case, you may be able to define the Endpoint gateway located in another site in the Endpoint login interfaces. However, normally you should not do that because it makes the Endpoint configuration more complicated. This might require customizing the login interface list for each set of Endpoints.

9.2.1 Deployment Considerations

When you deploy many Endpoints in your environment, to create the login interface list for each Endpoint, you will select one of the following ways:

- `-g` option
- select_gateway_policy

Each way has its own considerations.

9.2.1.1 Using the -g Option

This is the most typical way to create the login interfaces list for each Endpoint, however, if you deploy large numbers of Endpoints, it can be complicated and difficult because you need to specify the appropriate Endpoint gateways with the `-g` option for each Endpoint. When you plan the deployment, you should consider this, depending on the number of Endpoints that you will deploy and how many days you can spend. This method does not increase the Endpoint Manager’s load.

9.2.1.2 Using select_gateway_policy

This is the most powerful way to automate creating the login interfaces list for each Endpoint, especially for mass deployment. However, if you use the select_gateway_policy to create the login interfaces list for each Endpoint, you need to consider the following issues:

- Endpoint naming syntax
- Endpoint Manager’s load

To simplify the select_gateway_policy script, you must define the Endpoint naming syntax before deployment. In this naming syntax, the Endpoint name should correspond to the physical location of the Endpoint. This makes the select_gateway_policy simple and easy.

If you deploy large numbers of Endpoints this way, you should not deploy all Endpoints at the same time. Depending on the overhead involved in your select_gateway policy, you may be putting an extensive load on the Endpoint

Manager. You need to understand these considerations and plan for a phased deployment, if needed.

9.3 Useful Tools for Using TMA

In this section, we introduce the tools which we developed during the development of this redbook. These tools may help you to manage a large environment.

9.3.1 Endpoint Gateway Migration Tool

This tool allows you to migrate all Endpoints that have already logged into the Endpoint gateway (A) to the Endpoint gateway (B). This operation is useful when you need to shutdown the Endpoint gateway machine for maintenance and so on. Figure 124 on page 367 shows operations the tool can perform. As we mentioned before, in this case, the migration can be performed even if the Endpoint gateway (A) is down.

9.3.1.1 How to Use the Tool

Normally, the tool is used with two arguments as follows:

```
mig_tool to_gw from_gw
```

where

mig_tool The file name of the tool.

to_gw The label of the Endpoint gateway which will manage the Endpoints to be migrated.

from_gw The label of the Endpoint gateway which is managing the Endpoints to be migrated.

In this tool, you can specify the Endpoints which you would like to migrate to another Endpoint gateway in a configuration file (eplist.txt). Then you don’t need to specify the second argument (`form_gw` option).

```
#!/bin/sh
#
# mig script
#
#$1 : Gateway Label that migrate to
#$2 : Gateway Label that migrate from
#If you do not specify $2, this script read ’./eplist.txt’ file
#and migrate all EPs listed in that file.
#If you specify $2, this script create ’./eplist.txt’ file.
#
#set -x
EPLIST="./eplist.txt"

if [ "$1" = "$2" ]; then
```

```
    echo  "Invalid parm."
    exit 1
fi

GWTO=$1
GWFROM=$2

from_exists=0
to_exists=0
wgateway | while read li
do
    set $li
    if [ "$2" = "$GWFROM" ];then
        from_exists=1
    elif [ "$2" = "$GWTO" ]; then
        to_exists=1
    fi
done

if [ $to_exists = 0 ] ; then
    echo "Invalid gateway"
    exit 1
fi
if [ ! "$GWFROM" = "" ] && [ $from_exists = 0 ]; then
    echo "Invalid gateway"
    exit 1
fi

if [ "$GWFROM" = "" ]; then
    echo Migrate to $GWTO according to $EPLIST
else
    echo Migrate $GWFROM to $GWTO
fi
echo "Press ENTER to start:"
read ch

if [ "$GWFROM" = "" ]; then
    while read li
    do
        echo wep $li migrate $GWTO
        wep $li migrate $GWTO
#       wep $li status >/dev/null
    done < $EPLIST
else
    target=0
    rm -f $EPLIST
    wep ls | while read li
    do
        set $li
        if [ $1 = 'G' ];then
            if [ $3 = $GWFROM ];then
                target=1
            else
                target=0
            fi
        else
            if [ $target = 1 ]; then
                echo wep $2 migrate $GWTO
                wep $2 migrate $GWTO
#               wep $2 status >/dev/null
                echo $2 >> $EPLIST
            fi
        fi
    done
fi
```

9.3.2 Duplicate Endpoint Login Check Tool

The Endpoint duplicate login is one of the most typical and complicated problems with Endpoint installation. During a mass installation of Endpoints, conditions often cause this to occur. Therefore, it is very important to understand the duplicate login, how to detect it and how to fix it. In this section, we explain what is an Endpoint duplicate login and also introduce the tool to check for a duplicate login.

9.3.2.1 What Is Endpoint Duplicate Login?

Sometimes, we can see an Endpoint duplicate login. This means one Endpoint machine is registered several times into the Tivoli database. As a result, there are multiple Endpoint entries for a single Endpoint machine in the Tivoli database. You can confirm this by using the `wep` command, and more than one Endpoint label appears for a single Endpoint machine. The Endpoint duplicate login occurs in the following situations:

- The Endpoint broadcasts the login request and multiple Endpoint gateways process it.
- The Endpoint detects communication timeout before receiving acknowledgment from the Endpoint gateway. The Endpoint attempts to perform the login to another Endpoint gateway even if the former Endpoint gateway has processed the login request.
- The Endpoint's local configuration file (lcf.dat) is deleted without deleting the Endpoint object information from the Endpoint Manager. If you don't delete object information using the `wdelep` command, the new Endpoint object information is created at the next initial login of the Endpoint, then the Endpoint duplicate login occurs.

To detect the Endpoint duplicate login, we developed the following sample script. This sample detects the duplicate login and then asks you whether to delete the object or not.

```
#!/bin/sh
AUTOANS=""# y for all yes, n for all no, "" for interactive mode
TMP_EP=eplist.txt
TMP_IP=iplist.txt
TMP_DOWN=downep.txt
prt=print# ’echo’ for Windows
CONS=‘tty‘# Could not use stdin as console with ’here document’
#set -x
function ask_yesno
{
    echo $* "?"
    if [ "$AUTOANS" = "" ]; then
    read ch < $CONS
    elif [ "$AUTOANS" = "y" ]; then
        echo "y"
        ch="y"
    else
        echo "n"
        ch="n"
    fi
    if [ "$ch" = "y" ] || [ "$ch" = "Y" ]; then
        return 1
    else
        return 0
    fi
}
function ask_enter
{
    echo $*
    if [ "$AUTOANS" = "" ]; then
        read ch < $CONS
    fi
}

function chk_alive
{
    rm -f $TMP_DOWN
    grep $ip $TMP_EP|awk '{print $1}' | while read li
    do
# wep status command sometimes returns 'alive' for failure endpoint.
#wep $li status >/dev/null 2>&1
        wadminep $li view_version >/dev/null 2>&1
        if [ $? = 0 ]; then
            echo "$li UP"
        else
            echo "$li DOWN"
        # Log EP label for wdelep_ep function
            echo $li >> $TMP_DOWN
        fi
    done
}

#
function wdelep_ep
{
    cat $TMP_DOWN |xargs -t wdelep
}

echo "Creating Endpoint List"
wep ls|grep Endpoint|awk '{print $2}'|xargs -n1 -iLABEL  \
    sh "$prt -n \"LABEL \";wep LABEL|grep address|awk  '{print \$2}'" > $TMP_EP

echo "Creating IP Address list"
cat $TMP_EP|awk '{print $2}'|sort|uniq > $TMP_IP

echo "Checking Duplicate Login"
while read ip
do
    count=0
    epcount=`grep $ip $TMP_EP|wc -l`
    if (( $epcount > 1 ));then
        echo "Followings are duplicated:"
        grep $ip $TMP_EP|awk '{print $1}'
        echo "Checking existence"
        ping -c1 `echo $ip|awk -F'+' '{print $1}'` >/dev/null
        if [ $? = 0 ]; then
```

```
            ask_yesno "Would you like to check which one is alive now"
            if [ $? = 1 ]; then
                chk_alive $ip
                ask_yesno "Would you like to wdelep all of down endpoint"
                if [ $? = 1 ]; then
                    wdelep_ep $ip
                fi
            fi
        else
            ask_enter " Endpoint maybe down. Press Enter."
        fi
    fi
done < ${TMP_IP}

rm -f ${TMP_EP} ${TMP_IP} ${TMP_DOWN}
```

As we mentioned, this script detects the duplicate login and ask you whether to delete it or not. If you would like to delete it, you specify `y` as the answer. Then the script executes the `wadminep` command to check the actual status of the Endpoint (active or not) and if the Endpoint is not reachable, the script ask you again whether to delete or not. Then if you answer `y` to this question, the script executes the `wdelep` command to delete the entry of the Endpoint.

The AUTOANS variable enables you to operate in the non-interactive mode. Specify `AUTOANS="y"` to answer `y` to all of the questions or specify `AUTOANS="n"` to answer `n` to all questions. The following shows sample instructions of this tool:

```
Creating Endpoint List
Creating IP Address list
Checking Duplicate Login
Followings are duplicated:
salmon.353
salmon
salmon.351
Checking existence
Endpoint maybe down. Press Enter.
Followings are duplicated:
.374
.373
.372
Checking existence
Would you like to check which one is alive now ?
y
.374 DOWN
.373 DOWN
.372 DOWN
Would you like to wdelep all of down endpoint ?
y
wdelep .374 .373 .372
```

9.3.3 Endpoint Status Check Tool

In your management environment, if you are using an SNMP-based network management tool, such as NetView for AIX, this tool should be useful. The NetView for AIX performs status polling every five minutes, so that NetView can detect if the network interface is down. However, NetView cannot detect if the `lcfd` daemon is down, because the network interface can be available even if the `lcfd` daemon becomes unavailable. In this situation, the following tool is useful.

```
#!/bin/ksh

wep ls | awk ’/Endpoint/ {print $2}’ | while read li
do
wep $li status > /dev/null 2>&1
    if [ $? != 0 ];
    then
        wsnmptrap -h netview 1.3.6.1.4.1.2.6.900 6 1 \
        1.3.6.1.4.1.2.6.900.1 OctetString Endpoint_Down_$li
    fi
done
```

If you use the NetView for AIX, you can implement the following management system using this tool.

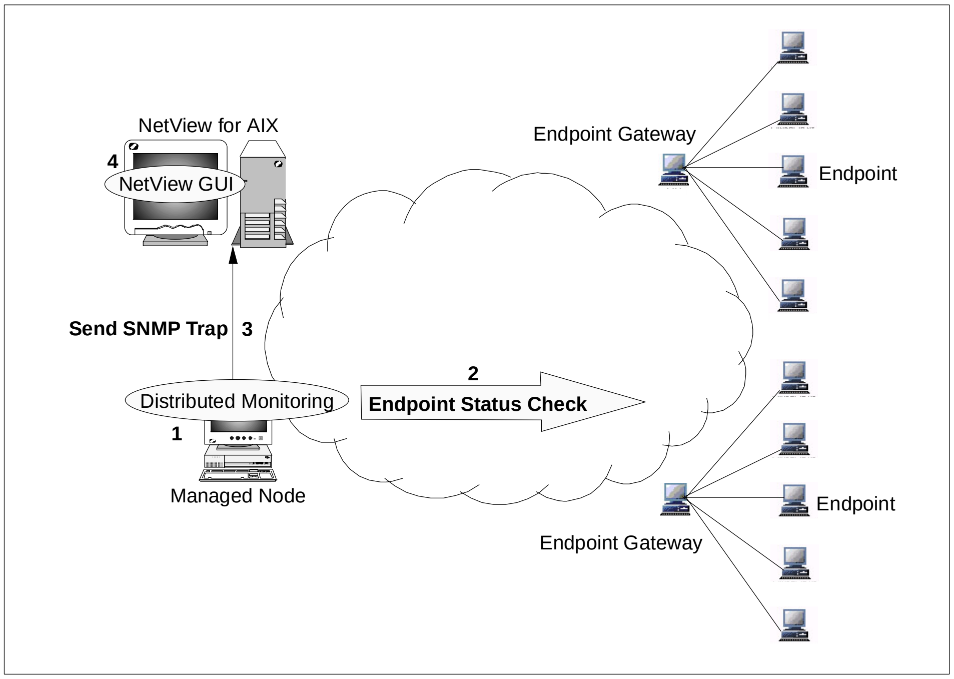


Figure 131. Endpoint Status Check with NetView for AIX

1. Create the Sentry Monitor which defines the Endpoint status tool in the run program field.
2. The Endpoint status tool should be executed at every interval and check the status of the Endpoint.
3. If the Endpoint status tool detects an unavailable Endpoint, then it sends the SNMP trap to NetView for AIX using the `wsnmptrap` command.
4. The NetView for AIX receives the SNMP trap and displays the event in the event console. You need to define the event configuration or ruleset of the NetView for AIX before the whole operation.

NetView for AIX provides an excellent network topology map on the display, so this kind of solution should be useful in a large environment.

Chapter 10. Tivoli Management Agent Performance Conisderations

Performance can be a key issue in enterprise systems management applications. The three-tiered management structure is designed to improve the performance and throughput in a Tivoli environment. However, to optimize the advantages of the three-tiered structure, you need to understand how to design your TMA environment and take actions to tune the performance. In this chapter, we introduce an overview of performance tuning in the TMA environment.

10.1 TMA Performance Tuning Strategy

To improve performance and throughput in the TMA environment, what should you do first? This is a very difficult question because it depends on many factors. The two most common approaches for improving performance in the Tivoli Management environment includes:

- Parameter Tuning (for each machine)
- Design Approach (for whole management system)

To understand parameter tuning, we can look at the following model which consists of six layers. As you can see, each layer depends on an upper or lower layer. In each layer, there are many ways to improve performance.

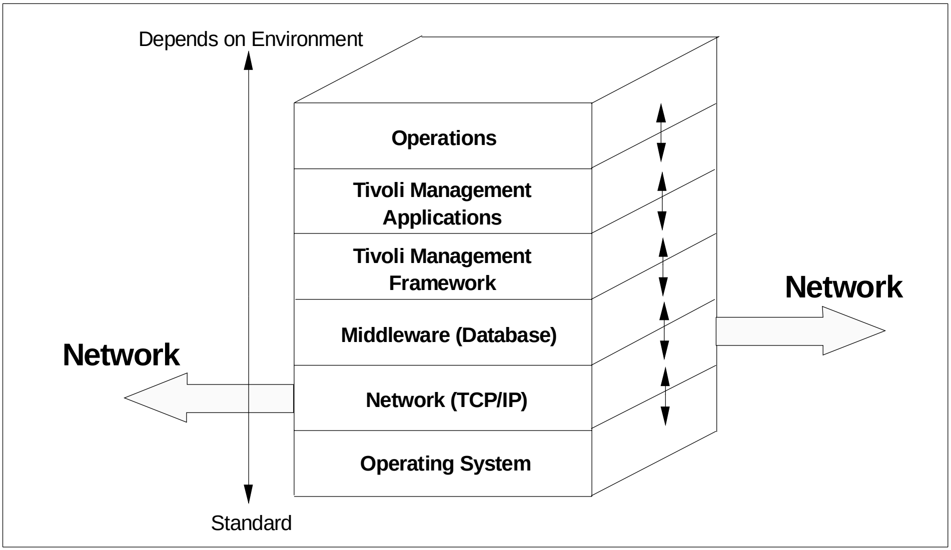


Figure 132. Performance Tuning Modeling

10.1.1 Operating System and Network Tuning

In real environments, of course, the relationship between each layer is complicated, so it is not in practice as simple as the above model. However, we can say that the Tivoli Management Framework runs on each operating system, for example AIX, Windows NT, or Windows98, and uses the network protocol (TCP/IP) provided by each system. Tuning the operating system and network is important for all distributed applications inlcuding Tivoli. For instance, for the RIM (RDBMS Interface Module) server performance tuning, the tuning of OS parameters, network parameters and RDBMS parameters is as important as tuning the RIM parameters themselves.

It is the same for TMA performance tuning. The system and network performance affects the TMA performance a lot. Network performance is really important in the Tivoli management environment. Network tuning is more standardized than the upper layer tuning because so many applications depend on it and the concepts are well understood by most network administrators.

10.1.2 TMA Tuning

After tuning the OS or network, to improve TMA performance, we need to consider tuning the Endpoint Manager, Endpoint gateway and Endpoint. This normally depends on each environment. The following list summarizes the some of the more common considerations:

- How many Endpoints are running under a single Endpoint gateway?
- The hardware specifications of the Endpoint Manager and Endpoint gateways (such as CPU or memory).
- The network design and speed.
- The TMR design.
- The applications running on the TMA (such as Software Distribution).
- Endpoint policy.

If you use Software Distribution, the MDist repeater tuning should be very important for performance and throughput. As you can see, it is very complicated. You have to understand the above dependencies and consider the most efficient way to improve TMA performance.

10.1.3 Tivoli Management Application Tuning

This depends on several environmental factors as well. In this book, we will not talk about tuning the Tivoli Management Applications, however, in most cases, how to optimize the Tivoli Management Applications is the same regardless if using Managed Nodes or TMAS Endpoints.

10.1.4 Operational Considerations

It is important to understand how Tivoli and the tivoli applications will be used and under what conditions. Understanding this, can help you to optimize the performance for your given environment.

For example, in a classroom system, all Endpoints might attempt to perform a login at the same time (when the class begins). This is very important information for the administrator. If you understand this situation before the TMA implementation, you can anticipate and possibly avoid problems

As another example, if you use Software Distribution, it is important to know how many files are distributed in a day, how many targets you have, and how large the files are. For example, if you distribute many large files, you should not necessarily send them to all systems at the same time.

Operational considerations may not improve the throughput dynamically, however, defining operational procedures based on the environment may help keep the throughput and performance within expectations even if the number of Endpoints increase.

10.2 TMR and Network Design Approach

In the previous section, we talked about performance tuning strategy by parameter tuning for each machine. However, in most environments, each system is connected via the network and working together. Therefore, the design of the whole management system is really important for performance tuning rather than tuning individual managed systems. The following figure represetns the interaction between each system.

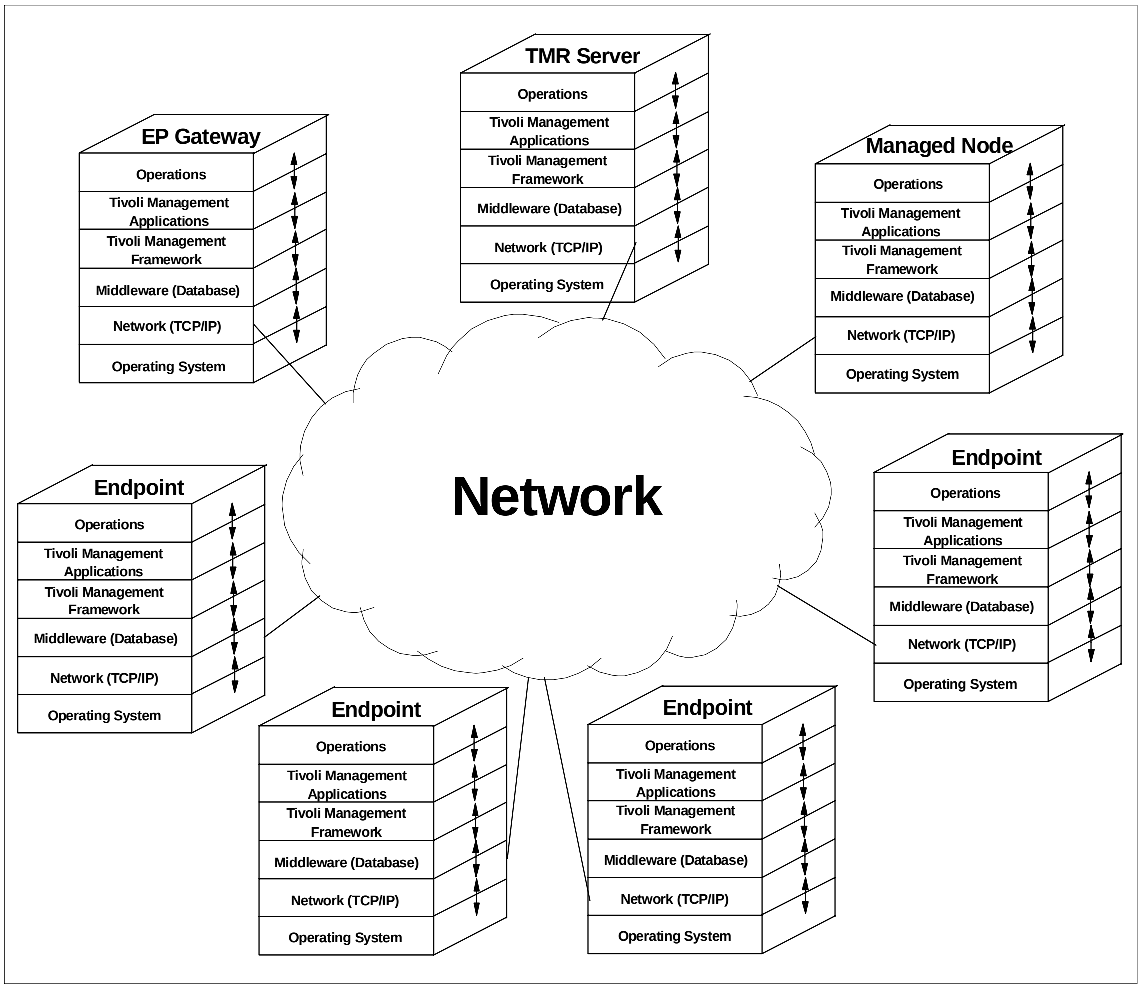


Figure 133. Interactions Between Each System

What you should consider next is how each box (system) can work together efficiently.

10.2.1 Design and Tuning

The design of the whole system is as important as tuning each system. Normally, the design of the management system would be considered in the early planning phase. This means that once you decide the design, it is very difficult to change it after the implementation. What about parameter tuning? Normally, it can be done as an on-going exercise after the implementation. The most important thing to remember is that it is much more difficult to change the design once you have begun the implement of the management system in a large environment.

10.2.2 Design Considerations

In Chapter 9, “Management Examples Using TMA” on page 359, we talked about the TMR design and management examples. Basically, this information should be useful for tuning performance and throughput as well.

The following should be considered in the design phase for improving the performance in a large environment:

- TMR Design.
- Network Design.
- How many Endpoints are managed in the TMR?
- MDist Repeater Allocation.
- RIM Server Allocation.
- TEC Server Allocation.

10.3 Understanding Parameters for Performance Tuning

In the TMA environment, there are some tuning parameters for each management resource. In this section, we introduce the parameters which affect the performance of the TMA environment.

10.3.1 Endpoint Gateway

You can configure the maximum number of concurrent jobs on the Endpoint gateway as follows. This depends on the environment, so be careful when you change it.

```
idlcall <gw_oid> _set_max_concurrent_jobs value
```

max_concurrent_jobs Sets the maximum number of concurrent jobs the Endpoint gateway can run. This setting, which controls the number of threads running in the Endpoint gateway, is the mechanism for controlling the resources (such as CPU and memory) the Endpoint gateway uses. The default setting for `max_concurrent_jobs` is 200.

A job consists of an Endpoint login, upcall or downcall. An MDist distribution consists of one job for the input stream and one job for each Endpoint it is being distributed to. Each job runs in its own thread. If the `max_concurrent_jobs` is exceeded, the job request goes on a wait queue. When there is a slot open, the Endpoint gateway starts the job as a thread. Hence, if the Endpoint gateway is busy, you don't get hard failures, but the quality of service suffers. If you notice that service is poor through a particular Endpoint gateway, you can create one or more additional Endpoint gateways in order to spread the load.

You can configure the network related parameters on the Endpoint gateway as well. It also depends on the environment and Tivoli Management Applications, so be careful when you change it. You can change this value using the `wrpt` command. This value is available for the Endpoint Manager as well.

net_load	Specifies the maximum amount of data (in KB per second) that the parameter will send to the network for each distribution.
max_conn	Specifies the maximum number of simultaneous parallel client connections initiated by the repeater during a distribution.

10.3.2 Endpoint Manager

You can define thenumber of concurrent jobs on the Endpoint Manager, as well. This depends on the environment, so be careful when you change it. You can change this value using the `idlattr` command, as follows. Refer to Chapter 7, "Advanced Knowledge of the TMA" on page 251 for more information.

```
idlattr -t -s 1588251808.1.517 max_jobs short value
```

max_jobs	Sets the maximum number of concurrent jobs on the Endpoint Manager. The default value is 20.

Note

In Version 3.6.1 of the Tivoli Management Framework, the max_jobs is not supported. Instead of max_jobs, the following parameters are supported by Version 3.6.1 of the Tivoli Management Framework.

max_install	Determines the number of maximum allow_install_policy scripts.
max_sgp	Governs the number of select_gateway_policy scripts.
max_after	Constructs the number of after_install_policy scripts.

These parameters control the number of respective policy scripts that the Endpoint Manager will run at the same time. All three default to 10, and thus at any given time, there are a maximum of 30 scripts that can run on the Endpoint Manager. These can be set using:

```
idlattr -t -s <epmgr_oid> <attr_name> short value
```

10.3.3 Operating System and Network

Tuning the OS and network, obviously is very operating system dependent. The following are very common parameters for system and network tuning.

Paging Space	You should customize this value. This is the most easy and steady way to improve performance.
nice value	Specifies the priority for each process. It is available on the AIX system. This value is dangerous, so you must refer to a manual of the AIX system if you change it.
mbuf	Specifies the maximum memory to be used for the network. It will be pinned in the real memory, so be careful when you change it. Before changing it, you should check the output of the `netstat -m` or `netstat -v` command.
tx_que_size	Specifies the transmit queue size of the adapter card. It is available on AIX systems.
rx_que_size	Specifies the receive queue size of the adapter card. It is available on AIX systems.

These are only typical parameters, so when you customize the system and network parameter, refer to the appropriate manual of the system.

10.4 Sample Configurations

In this section, we introduce a sample configuration for an Endpoint gateway and Endpoint. Again, this depends on the environment, so check your environment before implementing.

10.4.1 Endpoint Configuration

The following are the recommended configurations for the Endpoint:

log_threshold	Set level 0 (default). You can use the -d option as well.
dcast_disabled	Set disabled (bcast_disabled=1).
lcs.login_interfaces	Specifies at least two Endpoint gateways, hopefully more than three. Refer to Chapter 9, “Management Examples Using TMA” on page 359 for more information.

10.4.2 Endpoint Gateway Configuration

The configuration of the Endpoint gateway depends a great deal on the environment. The following is a very common configuration. We recommend you to configure as follows:

set_debug_level We recommend you set the debug level to 0. The debug level of the Endpoint gateway defines what information is logged into the $DBDIR/gatelog file. The debug level 0 means that it only logs errors of the Endpoint gateway into the gatelog file. This improves performance and ensures that the size of the gatelog file doesn’t become large quickly. To change the Endpoint gateway’s debug level from the CLI, you can use the following command:

```
wlookup -o -r Gateway -a | while read oid
do
    idlcall $oid _set_debug_level 0
done
```

You can use the `wgateway` command to set the debug level as well.

```
wgateway <gw_label> set_debug_level [0-9]
```

10.5 Improving Performance of TMA

We provided an overview of the performance tuning in the Tivoli Management environment, so in this section we will introduce hints and tips for improving performance.

10.5.1 Endpoint Policy Considerations

The Endpoint policy is a very useful way to manage Endpoints efficiently. However, it also expends resources to the Endpoint Manager and Endpoint gateways. Sometimes it may cause a performance problem, so that you should take care when using Endpoint policy.

We introduced the auto upgrade function of the TMA in Chapter 5, “Anatomy of TMA Behavior” on page 135. It allows us to upgrade the Endpoint software automatically. This should be a very useful function. However, to enable this function, upgrade.sh must be configured in the login_policy. This means that the auto upgrade function will be run every time the Endpoint logs into the Endpoint gateway even if the upgrade process does not occur. Moreover, the upgrade.sh includes the `awk` or `grep` command. The `awk` or `grep` commands are useful tools but they expend much more resource than other commands. If many Endpoints attempt to log into the Endpoint gateway at the same time, upgrade.sh will be executed for each Endpoint and the upgrade process will not occur on most of the Endpoints. This is really bad for performance tuning.

If you use the auto upgrade function, you should use it only when the Endpoint gateway is upgraded because then the contents under the $DBDIR/../bin/lcf_bundle directory should be upgraded as well. To modify the Endpoint policy, you can use the `wgeteppol` and `wputeppol` commands.

Auto upgrade is a typical case. The Endpoint policy can have an adverse affect on performance. To avoid this, you should take care of the following:

- Avoid using many `awk` or `grep` commands in Endpoint policy.
- Use a C program instead of the shell script, if possible, in the Endpoint policy.
- Don’t define an Endpoint policy if ti is not needed (like auto upgrade). It should be defined only when needed.

10.5.2 MDist Repeater and File Package Source Host

If you use Software Distribution, you have to take care of the configuration of the file package source host of Software Distribution. As you know, the file package source host still needs the full Managed Node function, so it should be configured on a full Managed Node. When you create a file package, the source host should be configured as an MDist repeater. The reason you should do this is for performance. If the source host is configured as an MDist repeater, the file package will be sent to the other MDist repeaters directly when the file package is distributed, but if the source host is not an MDist repeater, it will take another, less efficient route.

To create and configure the MDist repeater, you can use the `wrpt` command.

Appendix A. Endpoint Policy Argument Values

In this appendix, we show the argument values that the Enpoint policy returns for each platform.

A.1 Windows 95

The following are the argument values that the Endpoint policy returns on Windows 95.

```
1998/12/07 14:41:27: [allow_install_policy]
1998/12/07 14:41:27: The label of the ep machine: salmon.itsc.austin.ibm.com
1998/12/07 14:41:27: The object reference of the ep machine: OBJECT_NIL
1998/12/07 14:41:27: The architecture type of the ep machine: win95
1998/12/07 14:41:27: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/07 14:41:27: The ip address of the ep logging in.: 9.3.1.193+9494
1998/12/07 14:41:27: region: 1189622596
1998/12/07 14:41:27: dispatcher: 0
1998/12/07 14:41:27: version: 5
1998/12/07 14:41:27: LCF_LOGIN_STATUS=0
1998/12/07 14:41:29: Exitting ...
1998/12/07 14:41:30: [select_gateway_policy]
1998/12/07 14:41:30: The label of the ep machine: salmon.itsc.austin.ibm.com
1998/12/07 14:41:30: The object reference of the ep machine: OBJECT_NIL
1998/12/07 14:41:30: The architecture type of the ep machine: win95
1998/12/07 14:41:30: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/07 14:41:30: The ip address of the ep logging in.: 9.3.1.193+9494
1998/12/07 14:41:30: region: 1189622596
1998/12/07 14:41:30: dispatcher: 0
1998/12/07 14:41:30: version: 5
1998/12/07 14:41:30: LCF_LOGIN_STATUS=0
1998/12/07 14:41:32: Exitting ...
1998/12/07 14:41:33: [after_install_policy]
1998/12/07 14:41:34: The label of the ep machine: salmon.itsc.austin.ibm.com
1998/12/07 14:41:34: The object reference of the ep machine:
1189622596.301.508+#TMF_Endpoint::Endpoint#
1998/12/07 14:41:34: The architecture type of the ep machine: win95
1998/12/07 14:41:34: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/07 14:41:34: The ip address of the ep logging in.: 9.3.1.193+9494
1998/12/07 14:41:34: region: 1189622596
1998/12/07 14:41:34: dispatcher: 301
1998/12/07 14:41:34: version: 5
1998/12/07 14:41:34: LCF_LOGIN_STATUS=0
1998/12/07 14:41:36: Exitting ...
1998/12/07 14:41:47: [login_policy]
1998/12/07 14:41:47: The label of the ep machine: salmon.itsc.austin.ibm.com
1998/12/07 14:41:47: The object reference of the ep machine:
1189622596.301.508+#TMF_Endpoint::Endpoint#
1998/12/07 14:41:47: The architecture type of the ep machine: win95
1998/12/07 14:41:47: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/07 14:41:47: The ip address of the ep logging in.: 9.3.1.193+9494
1998/12/07 14:41:47: region: 1189622596
1998/12/07 14:41:47: dispatcher: 301
1998/12/07 14:41:47: version: 5
1998/12/07 14:41:47: LCF_LOGIN_STATUS=
1998/12/07 14:41:49: Exitting ...
```

A.2 Windows 98

The following are the argument values that the Endpoint policy returns on Windows 98.

```
1998/12/07 14:34:30: [allow_install_policy]
1998/12/07 14:34:30: The label of the ep machine: salmon
1998/12/07 14:34:30: The object reference of the ep machine: OBJECT_NIL
1998/12/07 14:34:30: The architecture type of the ep machine: win95
1998/12/07 14:34:30: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/07 14:34:30: The ip address of the ep logging in.: 9.3.1.193+9494
1998/12/07 14:34:30: region: 1189622596
1998/12/07 14:34:30: dispatcher: 0
1998/12/07 14:34:31: version: 5
1998/12/07 14:34:31: LCF_LOGIN_STATUS=0
1998/12/07 14:34:33: Exitting ...
1998/12/07 14:34:33: [select_gateway_policy]
1998/12/07 14:34:33: The label of the ep machine: salmon
1998/12/07 14:34:33: The object reference of the ep machine: OBJECT_NIL
1998/12/07 14:34:33: The architecture type of the ep machine: win95
1998/12/07 14:34:33: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/07 14:34:33: The ip address of the ep logging in.: 9.3.1.193+9494
1998/12/07 14:34:33: region: 1189622596
1998/12/07 14:34:33: dispatcher: 0
1998/12/07 14:34:33: version: 5
1998/12/07 14:34:33: LCF_LOGIN_STATUS=0
1998/12/07 14:34:35: Exitting ...
1998/12/07 14:34:38: [after_install_policy]
1998/12/07 14:34:38: The label of the ep machine: salmon
1998/12/07 14:34:38: The object reference of the ep machine:
1189622596.300.508+#TMF_Endpoint::Endpoint#
1998/12/07 14:34:38: The architecture type of the ep machine: win95
1998/12/07 14:34:38: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/07 14:34:38: The ip address of the ep logging in.: 9.3.1.193+9494
1998/12/07 14:34:38: region: 1189622596
1998/12/07 14:34:38: dispatcher: 300
1998/12/07 14:34:38: version: 5
1998/12/07 14:34:38: LCF_LOGIN_STATUS=0
1998/12/07 14:34:40: Exitting ...
1998/12/07 14:34:51: [login_policy]
1998/12/07 14:34:51: The label of the ep machine: salmon
1998/12/07 14:34:51: The object reference of the ep machine:
1189622596.300.508+#TMF_Endpoint::Endpoint#
1998/12/07 14:34:51: The architecture type of the ep machine: win95
1998/12/07 14:34:51: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/07 14:34:51: The ip address of the ep logging in.: 9.3.1.193+9494
1998/12/07 14:34:51: region: 1189622596
1998/12/07 14:34:51: dispatcher: 300
1998/12/07 14:34:52: version: 5
1998/12/07 14:34:52: LCF_LOGIN_STATUS=
1998/12/07 14:34:54: Exitting ...
```

A.3 Windows NT

The following shows the argument values that the Enpoint policy returns on Windows NT.

```
1998/12/07 13:49:50: [allow_install_policy]
1998/12/07 13:49:50: The label of the ep machine: salmon
1998/12/07 13:49:50: The object reference of the ep machine: OBJECT_NIL
1998/12/07 13:49:50: The architecture type of the ep machine: w32-ix86
1998/12/07 13:49:50: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/07 13:49:50: The ip address of the ep logging in.: 9.3.1.193+9494
1998/12/07 13:49:50: region: 1189622596
1998/12/07 13:49:50: dispatcher: 0
1998/12/07 13:49:50: version: 5
1998/12/07 13:49:50: LCF_LOGIN_STATUS=0
1998/12/07 13:49:52: Exitting ...
1998/12/07 13:49:52: [select_gateway_policy]
1998/12/07 13:49:52: The label of the ep machine: salmon
1998/12/07 13:49:52: The object reference of the ep machine: OBJECT_NIL
1998/12/07 13:49:52: The architecture type of the ep machine: w32-ix86
1998/12/07 13:49:52: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/07 13:49:52: The ip address of the ep logging in.: 9.3.1.193+9494
1998/12/07 13:49:52: region: 1189622596
1998/12/07 13:49:52: dispatcher: 0
1998/12/07 13:49:52: version: 5
1998/12/07 13:49:52: LCF_LOGIN_STATUS=0
1998/12/07 13:49:54: Exitting ...
1998/12/07 13:49:56: [after_install_policy]
1998/12/07 13:49:56: The label of the ep machine: salmon
1998/12/07 13:49:56: The object reference of the ep machine:
1189622596.299.508+#TMF_Endpoint::Endpoint#
1998/12/07 13:49:56: The architecture type of the ep machine: w32-ix86
1998/12/07 13:49:56: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/07 13:49:56: The ip address of the ep logging in.: 9.3.1.193+9494
1998/12/07 13:49:56: region: 1189622596
1998/12/07 13:49:56: dispatcher: 299
1998/12/07 13:49:56: version: 5
1998/12/07 13:49:56: LCF_LOGIN_STATUS=0
1998/12/07 13:49:58: Exitting ...
1998/12/07 13:50:09: [login_policy]
1998/12/07 13:50:09: The label of the ep machine: salmon
1998/12/07 13:50:09: The object reference of the ep machine:
1189622596.299.508+#TMF_Endpoint::Endpoint#
1998/12/07 13:50:09: The architecture type of the ep machine: w32-ix86
1998/12/07 13:50:09: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/07 13:50:09: The ip address of the ep logging in.: 9.3.1.193+9494
1998/12/07 13:50:09: region: 1189622596
1998/12/07 13:50:09: dispatcher: 299
1998/12/07 13:50:09: version: 5
1998/12/07 13:50:09: LCF_LOGIN_STATUS=
1998/12/07 13:50:11: Exitting ...
```

A.4 AIX V4.2

The following are the argument values that the Endpoint policy returns on AIX V4.2.

```
1998/12/10 13:21:09: [allow_install_policy]
1998/12/10 13:21:09: The label of the ep machine: bass
1998/12/10 13:21:09: The object reference of the ep machine: OBJECT_NIL
1998/12/10 13:21:09: The architecture type of the ep machine: aix4-r1
1998/12/10 13:21:09: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/10 13:21:09: The ip address of the ep logging in.: 9.3.1.209+9494
1998/12/10 13:21:09: region: 1189622596
1998/12/10 13:21:09: dispatcher: 0
1998/12/10 13:21:09: version: 5
1998/12/10 13:21:09: LCF_LOGIN_STATUS=0
1998/12/10 13:21:11: Exitting ...
1998/12/10 13:21:11: [select_gateway_policy]
1998/12/10 13:21:11: The label of the ep machine: bass
1998/12/10 13:21:11: The object reference of the ep machine: OBJECT_NIL
1998/12/10 13:21:11: The architecture type of the ep machine: aix4-r1
1998/12/10 13:21:11: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/10 13:21:12: The ip address of the ep logging in.: 9.3.1.209+9494
1998/12/10 13:21:12: region: 1189622596
1998/12/10 13:21:12: dispatcher: 0
1998/12/10 13:21:12: version: 5
1998/12/10 13:21:12: LCF_LOGIN_STATUS=0
1998/12/10 13:21:14: Exitting ...
1998/12/10 13:21:15: [after_install_policy]
1998/12/10 13:21:15: The label of the ep machine: bass
1998/12/10 13:21:15: The object reference of the ep machine:
1189622596.345.508+#TMF_Endpoint::Endpoint#
1998/12/10 13:21:15: The architecture type of the ep machine: aix4-r1
1998/12/10 13:21:15: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/10 13:21:15: The ip address of the ep logging in.: 9.3.1.209+9494
1998/12/10 13:21:15: region: 1189622596
1998/12/10 13:21:15: dispatcher: 345
1998/12/10 13:21:15: version: 5
1998/12/10 13:21:15: LCF_LOGIN_STATUS=0
1998/12/10 13:21:17: Exitting ...
1998/12/10 13:21:34: [login_policy]
1998/12/10 13:21:34: The label of the ep machine: bass
1998/12/10 13:21:34: The object reference of the ep machine:
1189622596.345.508+#TMF_Endpoint::Endpoint#
1998/12/10 13:21:34: The architecture type of the ep machine: aix4-r1
1998/12/10 13:21:34: The object reference of the gateway: 1189622596.2.19#TMF_Gateway::Gateway#
1998/12/10 13:21:34: The ip address of the ep logging in.: 9.3.1.209+9494
1998/12/10 13:21:34: region: 1189622596
1998/12/10 13:21:34: dispatcher: 345
1998/12/10 13:21:34: version: 5
1998/12/10 13:21:34: LCF_LOGIN_STATUS=
1998/12/10 13:21:36: Exitting ...
```

Appendix B. Making Batch Files

In this appendix, we show the making of the batch files that are used for the sample programs of the Endpoint applications.

B.1 Upcall Server (Platform)

The following batch file is used for the upcall server program at the make process.

```
#!/bin/sh
# Make script to compile upcall server
ClassName=Upcall
ProdName=LCF
OName=test
GWMain=upmeth
#
ROOT=c:/Tivoli
SRC=${ROOT}/src
#
# Install
#
function install() {
echo $1 $2
if [ ! -d $2 ]; then
mkdir -p $2
fi
rm -f $2/$1
install.sh -m 0644 $1 $2/$1
}

#
# Compile
#
export CCOPT="-MD -DWIN32 \
-DSTDC_SRC_COMPATIBLE \
-I. \
-I${SRC}/export/include/w32-ix86 \
-I${SRC}/export/include/generic  \
-I${ROOT}/include/w32-ix86 \
-I${ROOT}/include/generic"

function cl_source() {
rm -f $1.obj
cl $CCOPT -c $2$1.c -Fo$1.obj
}

#
# Main
#
if [ ! -d ../w32-ix86 ]; then
mkdir -p ../w32-ix86
fi
cd ../../common
# create idl/ist/prog/imp files
sgen < ${ClassName}.gen

cd ../platform/w32-ix86
```

```
# IDL compile
tidlc  -I. \
-I${ROOT}/include/w32-ix86 \
-I${ROOT}/include/generic \
../../common/${ClassName}.ist

install ${ClassName}_defs.h ${SRC}/export/include/w32-ix86/tivoli
install ${ClassName}.h ${SRC}/export/include/w32-ix86/tivoli
install t_${ClassName}.h ${SRC}/export/include/w32-ix86/tivoli
install ${ClassName}_aux.h ${SRC}/export/include/w32-ix86/tivoli
install ${ClassName}_imp.h ${SRC}/export/include/w32-ix86/tivoli
install t_${ClassName}_imp.h ${SRC}/export/include/w32-ix86/tivoli
#install ${ClassName}.idl ${SRC}/export/include/w32-ix86
#install ${ClassName}.ist ${SRC}/export/include/w32-ix86
#install ${ClassName}.imp ${SRC}/export/include/w32-ix86
#install ${ClassName}.prog ${SRC}/export/include/w32-ix86

cl_source $GWMain ../src/./
cl_source ${ClassName}_main
cl_source ${ClassName}_main_skel
cl_source ${ClassName}_aux
cl_source ${ClassName}_imp
cl_source t_${ClassName}_imp
cl_source t_${ClassName}_main_skel

export LKOPT="/LIBPATH:${ROOT}/lib/w32-ix86
-subsystem:console
libtas.a libtmfimp.a libtmf.a libms.a
libdes.a libthreads.a"

link $LKOPT \
-out:$GWMain.exe $GWMain.obj \
${ClassName}_aux.obj ${ClassName}_imp.obj ${ClassName}_main.obj \
t_${ClassName}_imp.obj ${ClassName}_main_skel.obj t_${ClassName}_main_skel.obj

install $GWMain.exe ${SRC}/export/bin/w32-ix86/${ProdName}/${OName}
install ${ClassName}.cfg ${SRC}/export/cfg/w32-ix86/tivoli
install ${ClassName}_ir.tar ${SRC}/export/cfg/w32-ix86/tivoli
install ${ClassName}_ist.tar ${SRC}/export/cfg/w32-ix86/tivoli
cd ../src
install ${ClassName}.init ${SRC}/export/cfg/w32-ix86/tivoli
```

B.2 Upcall Client (Endpoint)

The following batch file is used for the upcall client program.

```
#!/bin/sh
# make script to compile upcall client
#
ClassName=Upcall
#
ROOT=c:/Tivoli
SRC=${ROOT}/src
LCFROOT=${ROOT}/bin/lcf_bundle
#
ProdName=LCF
OName=test
#
# Install
#
function install() {
```

```
if [ ! -d $2 ]; then
mkdir -p $2
fi
rm -f $2/$1
install.sh -m 0644 $1 $2/$1
}

#
# Compile
#
export CCOPT="-MD -DWIN32 -DENDPOINT_BUILD -DSTDC_SRC_COMPATIBLE -DPC \
-I. \
-I../cross/. \
-I${LCFROOT}/include/w32-ix86 \
-I${LCFROOT}/include/generic \
-I${ROOT}/include/w32-ix86"
function cl_source() {
rm -f $1.obj
cl $CCOPT -c $2$1.c -Fo$1.obj
}

#
# Main
#
if [ ! -d ../w32-ix86 ]; then
mkdir -p ../w32-ix86
fi
if [ ! -d ../cross ]; then
mkdir -p ../cross
fi

# IDL compile
cd ../cross
# Set PATH to ADE for Endpoint
export PATH="c:/tivoli/bin/lcf_bundle/bin/w32-ix86/ade;$PATH"
${LCFROOT}/bin/w32-ix86/ade/ltid \
-DENDPOINT_BUILD \
-I. \
-I${SRC}/export/include/w32-ix86 \
-I${SRC}/export/include/generic  \
-I${LCFROOT}/include/w32-ix86 \
-p ../../common/${ClassName}.ist

cd ../w32-ix86
cl_source ${ClassName} ../src/./
cl_source ${ClassName}_aux ../cross/./
cl_source t_${ClassName}_stub ../cross/./

#msvcrt.lib kernel32.lib advapi32.lib netapi32.lib user32.lib
#oldnames.lib libmrt.a libdes.a libcpl.a wsock32.lib wsock32.lib advapi32.lib
export LKOPT="/LIBPATH:${LCFROOT}/lib/w32-ix86 \
-subsystem:console \
libmrt.a libdes.a libcpl.a"

link $LKOPT -out:${ClassName}.exe \
${ClassName}.obj \
${ClassName}_aux.obj \
t_${ClassName}_stub

cp ${ClassName}.exe ${ClassName}
install ${ClassName} ${SRC}/export/bin/lcf_bundle/bin/w32-ix86/${ProdName}/${OName}
```

B.3 Downcall Client (Platform)

The following batch file is used for the downcall client program at the make process.

```
#!/bin/sh
# Make script to compile downcall client
#
ClassName=Downcall
GWMain=dsmain
#
ROOT=c:/Tivoli
SRC=${ROOT}/src
#
# Install
#
function install() {
echo $1 $2
if [ ! -d $2 ]; then
mkdir -p $2
fi
rm -f $2/$1
install.sh -m 0644 $1 $2/$1
}

#
# Compile
#
export CCOPT="-MD -DWIN32 \
-DSTDC_SRC_COMPATIBLE \
-I. \
-I${SRC}/export/include/w32-ix86 \
-I${SRC}/export/include/generic  \
-I${ROOT}/include/w32-ix86 \
-I${ROOT}/include/generic"

function cl_source() {
rm -f $1.obj
cl $CCOPT -c $2$1.c -Fo$1.obj
}

#
# Main
#
if [ ! -d ../w32-ix86 ]; then
mkdir -p ../w32-ix86
fi
cd ../../common
# create idl/ist/prog/imp files
sgen < ${ClassName}.gen

cd ../platform/w32-ix86
# IDL compile
tidlc  -I. \
-I${ROOT}/include/w32-ix86 \
-I${ROOT}/include/generic \
../../common/${ClassName}.ist

install ${ClassName}_defs.h ${SRC}/export/include/w32-ix86/tivoli
install ${ClassName}.h ${SRC}/export/include/w32-ix86/tivoli
install t_${ClassName}.h ${SRC}/export/include/w32-ix86/tivoli
install ${ClassName}_aux.h ${SRC}/export/include/w32-ix86/tivoli
```

```
#install ${ClassName}_imp.h ${SRC}/export/include/w32-ix86/tivoli
#install t_${ClassName}_imp.h ${SRC}/export/include/w32-ix86/tivoli
#install ${ClassName}.idl ${SRC}/export/include/w32-ix86
#install ${ClassName}.ist ${SRC}/export/include/w32-ix86
#install ${ClassName}.imp ${SRC}/export/include/w32-ix86
#install ${ClassName}.prog ${SRC}/export/include/w32-ix86

cl_source $GWMain ../src/./
cl_source ${ClassName}_aux
cl_source t_${ClassName}_stub

export LKOPT="/LIBPATH:${ROOT}/lib/w32-ix86
-subsystem:console
libtas.a libtmfimp.a libtmf.a libms.a
libdes.a libthreads.a"

link $LKOPT \
-out:$GWMain.exe $GWMain.obj \
${ClassName}_aux.obj t_${ClassName}_stub.obj \

install $GWMain.exe ${SRC}/export/bin/w32-ix86/bin
install ${ClassName}.cfg ${SRC}/export/cfg/w32-ix86/tivoli
install ${ClassName}_ir.tar ${SRC}/export/cfg/w32-ix86/tivoli
install ${ClassName}_ist.tar ${SRC}/export/cfg/w32-ix86/tivoli
cd ../src
install ${ClassName}.init ${SRC}/export/cfg/w32-ix86/tivoli
```

B.4 Downcall Server (Endpoint)

The following batch file is used for the downcall server program during the make process.

```
#!/bin/sh
# Make script to compile downcall serever
#
ClassName=Downcall
#
ROOT=c:/Tivoli
SRC=${ROOT}/src
LCFROOT=${ROOT}/bin/lcf_bundle
#
ProdName=LCF
OName=test
#
# Install
#
function install() {
if [ ! -d $2 ]; then
mkdir -p $2
fi
rm -f $2/$1
install.sh -m 0644 $1 $2/$1
}

#
# Compile
#
export CCOPT="-MD -DWIN32 -DENDPOINT_BUILD -DSTDC_SRC_COMPATIBLE -DPC \
-I. \
-I../cross/. \
-I${LCFROOT}/include/w32-ix86 \
-I${LCFROOT}/include/generic \
-I${ROOT}/include/w32-ix86"
function cl_source() {
rm -f $1.obj
cl $CCOPT -c $2$1.c -Fo$1.obj
}

#
# Main
#
if [ ! -d ../w32-ix86 ]; then
mkdir -p ../w32-ix86
fi
if [ ! -d ../cross ]; then
mkdir -p ../cross
fi

# IDL compile
cd ../cross
# Set PATH to ADE for Endpoint
export PATH="c:/tivoli/bin/lcf_bundle/bin/w32-ix86/ade;$PATH"
${LCFROOT}/bin/w32-ix86/ade/ltid \
-DENDPOINT_BUILD \
-I. \
-I${SRC}/export/include/w32-ix86 \
-I${SRC}/export/include/generic  \
-I${LCFROOT}/include/w32-ix86 \
-p ../../common/${ClassName}.ist

cd ../w32-ix86
cl_source ${ClassName} ../src/./
cl_source ${ClassName}_aux ../cross/./
cl_source ${ClassName}_main_skel ../cross/./
cl_source ${ClassName}_main ../cross/./
#cl_source ${ClassName}_imp ../cross/./
#cl_source t_${ClassName}_imp ../cross/./
#cl_source t_${ClassName}_main_skel ../cross/./

export LKOPT="/LIBPATH:${LCFROOT}/lib/w32-ix86 \
-subsystem:console \
msvcrt.lib kernel32.lib advapi32.lib netapi32.lib user32.lib \
oldnames.lib libmrt.a libdes.a libcpl.a wsock32.lib wsock32.lib advapi32.lib"
link $LKOPT -out:${ClassName}.exe \
${ClassName}.obj \
${ClassName}_aux.obj \
${ClassName}_main_skel.obj \
${ClassName}_main.obj
#${ClassName}_imp.obj \
#t_${ClassName}_imp.obj \
#t_${ClassName}_main_skel.obj \

cp ${ClassName}.exe ${ClassName}
install ${ClassName} ${SRC}/export/bin/lcf_bundle/bin/w32-ix86/${ProdName}/${OName}
```

Appendix C. Special Notices

This publication is intended to help technical users and customers of Tivoli products to understand more about the Tivoli Management Framework. The information in this publication is not intended as the specification of any programming interfaces that are provided by the Tivoli Management Framework. See the PUBLICATIONS section of the IBM Programming Announcement for the Tivoli Management Framework for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have

Appendix D. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

D.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see "How to Get ITSO Redbooks" on page 413.

D.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	**Subscription Number**	**Collection Kit Number**
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at `http://www.redbooks.ibm.com/`.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Redbooks Web Site on the World Wide Web**

  `http://w3.itso.ibm.com/`

- **PUBORDER** – to order hardcopies in the United States
- **Tools Disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLCAT REDPRINT
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type the following command:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1998
  ```

- **REDBOOKS Category on INEWS**
- **Online** – send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** – send orders to:

	IBMMAIL	**Internet**
In United States	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** – send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** – send orders to:

United States (toll free)	1-800-445-9269
Canada	1-800-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 408 256 5422 (Outside USA)** – ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	**Order Number**	**Quantity**

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

List of Abbreviations

APA	All Points Addressable
CORBA	Common Object Request Broker Architecture
DHCP	Dynamic Host Configuration Protocol
IBM	International Business Machines Corporation
ITSO	International Technical Support Organization
IR	Installation Repository
LCF	Lightweight Client Framework
LPP	Licensed Program Process
NAT	Network Address Translation
OMG	Object Management Group
TMA	Tivoli Management Agent
TMR	Tivoli Management Region
TNR	Tivoli Name Registry
TRIP	Tivoli Remote Execution Service

Index

ITSO Redbook Evaluation

All About Tivoli Management Agents
SG24-5134-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
_ Customer _ Business Partner _ Solution Developer _ IBM employee
_ None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction __________

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

What other redbooks would you like to see published?

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

SG24-5134-00
Printed in the U.S.A.

All About Tivoli Management Agents

SG24-5134-00