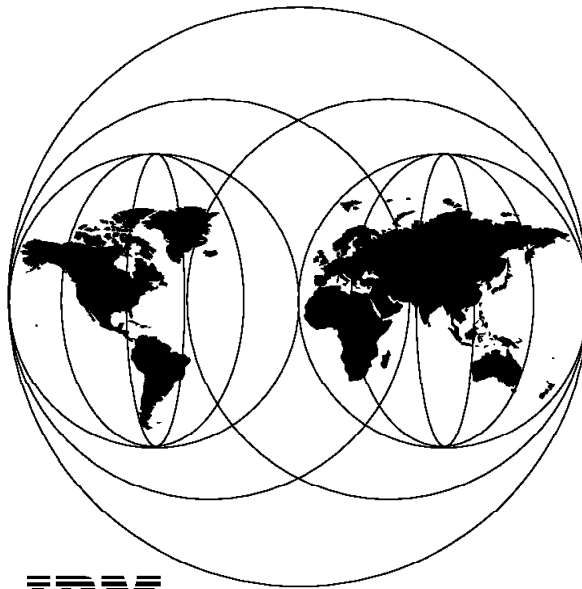


SG24-2025-00

**TME 10 Framework Version 3.2:  
An Introduction to the Lightweight Client Framework**

October 1997



**IBM**

**International Technical Support Organization  
Austin Center**



SG24-2025-00

International Technical Support Organization

**TME 10 Framework Version 3.2:  
An Introduction to the Lightweight Client Framework**

October 1997



**Take Note!**

Before using this information and the product it supports, be sure to read the general information in Appendix A, "Special Notices" on page 153.

**First Edition (October 1997)**

This edition applies to Version 3, Release 2 of the TME 10 Framework.

Comments may be addressed to:

IBM Corporation, International Technical Support Organization  
Dept. JN9B Building 045 Internal Zip 2834  
11400 Burnet Road  
Austin, Texas 78758-3493

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1997. All rights reserved.**

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

---

## Contents

<b>Figures</b> .....	vii
<b>Tables</b> .....	ix
<b>Preface</b> .....	xi
The Team That Wrote This Redbook .....	xi
Comments Welcome .....	xiii
<b>Chapter 1. Introduction</b> .....	1
1.1 TME 10 Overview .....	2
1.1.1 Operating System Independence .....	2
1.1.2 Distributed Object Framework .....	3
1.1.3 TME 10 Framework's Distributed Database .....	6
1.1.4 Basic Management Functions .....	7
1.1.5 Installation Routines .....	8
1.1.6 Command Interface .....	8
1.1.7 TME 10 Desktop (Graphical User Interface) .....	9
1.2 Tivoli Management Regions .....	9
1.2.1 Interconnecting TMRs .....	11
1.3 TME 10 Version 3.1 Limitations .....	11
1.4 Summary .....	12
<b>Chapter 2. TME 10 Framework V3.2 - What's New?</b> .....	13
2.1 Lightweight Client Framework .....	13
2.1.1 LCF Basics .....	13
2.1.2 How LCF Changes the TME 10 Topology .....	17
2.1.3 LCF and the TME 10 Desktop .....	17
2.1.4 LCF and the Command Line Interface .....	18
2.1.5 LCF Summary .....	19
2.2 RDBMS Interface Module - RIM .....	20
2.2.1 Summary .....	21
2.3 DynaText Document Viewer .....	21
2.4 Web Interface .....	22
2.5 Summary .....	22
<b>Chapter 3. Planning for TME 3.2</b> .....	23
3.1 Prerequisites .....	23
3.1.1 Operating Systems .....	24
3.1.2 Hardware Requirements .....	25
3.1.3 Server and Client Configuration .....	28
3.2 Network Requirements and Considerations .....	31

3.2.1	Line Throughput	31
3.2.2	DNS (Domain Name System)	31
3.2.3	DHCP (Dynamic Host Configuration Protocol)	32
3.2.4	DDNS (Dynamic DNS)	32
3.3	Tivoli Management Regions	33
3.4	Endpoint Connection Process	34
3.4.1	How an Endpoint Connects	34
3.4.2	Sequence of an Endpoint Connection	35
3.4.3	Limiting Endpoint Broadcasting	36
3.5	General LCF Planning Considerations	37
3.6	Summary	39
<b>Chapter 4. LCF Installation</b>		<b>41</b>
4.1	Endpoint Managers	41
4.1.1	Creating an Endpoint Manager	41
4.2	Endpoint Gateways	43
4.2.1	Creating an Endpoint Gateway - Command Line Interface	44
4.2.2	Creating an Endpoint Gateway - GUI	44
4.3	Endpoints	47
4.3.1	Default Endpoint Login Process	49
4.3.2	Creating a Windows NT Endpoint Using InstallShield	49
4.3.3	Installing Windows NT Endpoints from Logon Scripts	53
4.3.4	Installing UNIX Endpoints with winstlcf	55
4.4	Endpoint Data Files	55
4.5	Summary	60
<b>Chapter 5. Configuring the LCF Environment</b>		<b>61</b>
5.1	Controlling an Endpoint Login	61
5.2	Controlling an Endpoint Login Through Configuration Files	62
5.3	Controlling an Endpoint Login Through Policies	64
5.3.1	allow_install_policy	64
5.3.2	select_gateway_policy	66
5.3.3	after_install_policy	68
5.3.4	login_policy	70
5.3.5	Applying Endpoint Policies	71
5.4	Summary	73
<b>Chapter 6. TME 10 Framework Web Interface</b>		<b>75</b>
6.1	Accessing the TMR Server	75
6.1.1	Starting and Stopping the HTTP Daemon	77
6.1.2	Accessing the TMR Server	77
6.1.3	Security	77
6.1.4	Accessing Managed Nodes	77
6.1.5	Adding Your Own Information	78

6.1.6	Logs	78
6.2	LCF Endpoint Web Server	78
6.2.1	Accessing the LCF Daemon Status Page	79
6.2.2	Logfile Page	81
6.2.3	Method Cache Page	82
6.2.4	Usage Statistics Page	82
6.2.5	Configuration Settings Page	83
6.2.6	Trace Log Page	84
6.2.7	Network Address Configuration Page	85
6.3	Summary	87
<b>Chapter 7. Migration from TME 3.1 to TME 3.2</b>		89
7.1	Reasons for Migrating	89
7.2	Migration Process Overview	91
7.3	Migration Scenarios	93
7.3.2	Add a Gateway to a Managed Node	104
7.4	Summary	105
<b>Chapter 8. Application Development Considerations</b>		107
8.1	Application Design	107
8.1.1	TME Methods	107
8.1.2	Components of an LCF Application	112
8.2	The LCF programming Environment	114
8.3	Application Runtime Library	115
8.3.1	Memory Management	116
8.3.2	Distributed Exceptions	117
8.3.3	Sequence Manipulations	118
8.3.4	File System Input/Output	119
8.3.5	Logging Functions	119
8.3.6	ADR Marshalling Functions	120
8.4	The Common Porting Layer Runtime Library	121
8.5	Summary	124
<b>Chapter 9. Useful Tips and Scripts</b>		125
9.1	Backup and Recovery	125
9.1.1	Backup Process	125
9.1.2	Common Backup Issues	126
9.1.3	Run wchkdb Before Any Backup	128
9.2	Removing Managed Nodes	128
9.3	Transfer TME 10 CD Images to Disk	129
9.3.1	Using file0.tar	129
9.4	Troubleshooting Client Installation	130
9.5	Miscellaneous Scripts	131
9.5.1	Finding out Who's Running the TME 10 Desktop	131

9.5.2 Deleting a Managed Node . . . . .	131
9.5.3 Backup the TMR . . . . .	132
9.5.4 Installing a Full TMR . . . . .	133
9.6 Summary . . . . .	149
<b>Chapter 10. In Conclusion . . . . .</b>	<b>151</b>
<b>Appendix A. Special Notices . . . . .</b>	<b>153</b>
<b>Appendix B. Related Publications . . . . .</b>	<b>155</b>
B.1 International Technical Support Organization Publications . . . . .	155
B.2 Redbooks on CD-ROMs . . . . .	155
B.3 Other Publications . . . . .	155
<b>How to Get ITSO Redbooks . . . . .</b>	<b>157</b>
How IBM Employees Can Get ITSO Redbooks . . . . .	157
How Customers Can Get ITSO Redbooks . . . . .	158
IBM Redbook Order Form . . . . .	159
<b>Index . . . . .</b>	<b>161</b>
<b>ITSO Redbook Evaluation . . . . .</b>	<b>163</b>



---

## Figures

1.	Role of the ORB in a TME Environment	5
2.	TME's Distributed Database	7
3.	TME 3.1 Architecture	10
4.	TMR Utilizing LCF	16
5.	New Endpoint Manager Icon on TME Desktop	18
6.	Initial Endpoint Connection	34
7.	TME 10 Desktop with Endpoint Manager	42
8.	Output of wls Showing Endpoint Manager	42
9.	Endpoint Manager Immediately After Installation	43
10.	Gateway Installation	45
11.	Create Gateway Dialog	46
12.	Gateway List	47
13.	Initial Endpoint Installation Panel	50
14.	Endpoint Installation Parameters Panel	51
15.	Endpoint Installation Status Panel	52
16.	Gateway List after Multiple Endpoints are Installed	53
17.	LCFD.LOG File After Installation	57
18.	LAST.CFG File	59
19.	Sample lcf.cfg File	63
20.	Default allow_install_policy Script	72
21.	Primary Web Page on a TMR Server	76
22.	LCF Endpoint's Home Page	80
23.	LCF Endpoint's Logfile Page	81
24.	LCF Endpoint's Method Cache Page	82
25.	LCF Endpoint's Usage Statistics Page	83
26.	LCF Endpoint's Configuration Settings Page	84
27.	LCF Endpoint's Trace Log Page	85
28.	LCF Endpoint's Configuration Page	86
29.	Two Interconnected TMRs without LCF	94
30.	Single TMR with LCF	95
31.	Upgrade to 3.2 - Start from Desktop	97
32.	Upgrade to 3.2 - Install Patch - Error Message	98
33.	Upgrade to 3.2 - Install Patch	99
34.	Upgrade to 3.2 - Patch Install	100
35.	Upgrade to 3.2 - Patch Install (cont'd)	101
36.	Upgrade to 3.2 - Patch Install (cont'd)	101
37.	Upgrade to 3.2 - Patch Install (finished)	102
38.	TME Desktop after Upgrading to 3.2	103



---

## Tables

1.	TME 10 Framework Operating System Support	25
2.	Swap Space & Process Slots	26
3.	Disk Space Requirement Matrix	26
4.	LCF Release 3.2	93
5.	Memory Management Functions	116
6.	Distributed Exceptions - Available Macros	117
7.	Distributed Exceptions - Available Variable Argument Exception Functions	118
8.	Sequence Manipulations - Available Sequences Macros	118
9.	File system input/output - Available Functions	119
10.	Logging Functions - Available functions	120



---

## Preface

The TME 10 Framework implements a powerful architecture that enables cross platform, network-wide management of your network computing environment. Through a common set of interfaces, applications use this framework to allow you to manage the variety of platforms in your environment while providing operating system transparency.

This redbook documents the new features and capabilities introduced in Version 3.2 of Tivoli's TME 10 Framework. The most significant of these new features is an extension to the TME 10 architecture known as the Lightweight Client Framework (LCF). LCF brings the power of TME 10 to a wider array of computing platforms, while drastically reducing the footprint required on the managed system. In addition, the LCF architecture greatly increases the scalability of TME 10, so that it can be used to manage environments from workgroups to extremely large enterprise networks.

The majority of this redbook is dedicated to describing LCF, and how to plan for, install and configure its components. We also discuss migration considerations from previous versions of the TME 10 Framework. For those of you who will be writing applications to take advantage of LCF, we also include a chapter discussing application development considerations.

Although you won't be able to reap all of the benefits of LCF until LCF-enabled applications become available (which will occur in the near future), by reading this redbook you can become familiar with the LCF architecture and the changes that its new scalability and client features will allow you to implement in your managed environment. Read this book and start planning for your TME 10 implementation today!

---

## 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.

**Bart Jacob** is a Senior Software Engineer at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM classes worldwide on a variety of topics including Tivoli's TME 10 products. Bart has been with the ITSO for eight years and has extensive experience with systems management, object-oriented technologies, and data communications.

**Armand Adriano** is an I.T. Specialist in IBM Canada. He belongs to the Systems Management and Networking Services group of Western Canada

and is responsible for providing Tivoli professional services to customers. He has worked in IBM for seven years, six of that performing RS/6000 and AIX technical support.

**Francesco Fabrizi** is an Advisory I.T. Specialist with IBM Italy. He has six years of experience in the development and design of System Management applications. He has worked at IBM for seven years, six of that spent in the Rome Networking System Laboratory. His areas of expertise include Object Oriented technology (designing/coding) on INTEL and Unix-based platforms.

**Paul Gerhard Morlok** is a Senior Associate System Engineer at the IBM Boeblingen Programming Laboratory, Germany. He has eight years of experience in System Management as a quality engineer. Within the OS/390 Performance Management Development group, he is responsible for planning, designing, and implementing complex component and system test scenarios. His major focus currently is on system management applications integrating the OS/2, AIX and OS/390 platforms.

**Renata Rossi** is an I.T.Specialist System Engineer in Italy. She has eleven years of experience in Technical Support Groups, with five on AIX Communications Products, Systemview and Netview. Since the merger of IBM and Tivoli, she has been working as a TME 10 Availability Specialist.

The authors would like to thank the entire the TME 10 Framework team at Tivoli for their invaluable contributions and support for this project. Special thanks goes to:

John Michael Adams  
Ann Bishop  
Sally Derrick  
Jerry Frain  
Zafir Gan  
Russell Hill  
Mike McDaniels  
Todd Praisner  
Francis Sullivan  
Rob Tulloh  
Evan Watkins  
Donna Wilkins

---

## 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 163 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Home Pages at the following URLs:

For Internet users <http://www.redbooks.ibm.com>

For IBM Intranet users <http://w3.itso.ibm.com/redbooks>

- Send us a note at the following address:

[redbook@vnet.ibm.com](mailto:redbook@vnet.ibm.com)





---

## Chapter 1. Introduction

Tivoli's TME 10 Framework provides a set of common services and facilities to enable powerful systems management applications. This framework provides value to developers who take advantage of the services and facilities that hide the complexity of the networked environment and allow 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 by providing common user interface elements and hiding differences in the operating environment of managed systems.

Tivoli's 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 TME 10 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 overtime to meet their changing requirements.

Version 3.2 of the TME 10 Framework introduces major new extensions to the framework's architecture. These new extensions include the Lightweight Client Framework (LCF). This new version of the framework is a testament to both the stability and extensibility of the TME 10 architecture. Version 3.2 of the TME 10 Framework can be installed and configured the same as previous versions and current applications will continue to run. However, by utilizing the LCF extensions to the framework architecture, applications can evolve to meet scalability and footprint requirements not necessarily met by previous versions of the framework.

This redbook describes the new features provided by Version 3.2 of the TME 10 Framework including LCF. If you are unfamiliar with the TME 10, architecture this introductory chapter will introduce you to the key concepts and components. In addition, it describes some of the limitations which are now being lifted by the introduction of LCF. If you are familiar with previous versions of the TME 10 Framework you may want to skip ahead to Chapter 2, "TME 10 Framework V3.2 - What's New?" on page 13.

---

## 1.1 TME 10 Overview

TME 10 is the brand given to a set of products that provide systems management capabilities across multiple platforms.

TME is an acronym for Tivoli Management Environment. The Tivoli Management Environment includes a structured framework, TME 10 Framework, that provides core services and capabilities for management applications. TME 10 also includes a growing set of applications which take advantage of the framework.

From a high level view, the TME 10 Framework can be described as containing the following sets of services:

- Operating system interfaces/encapsulation
- Distributed object framework
- Distributed database
- Basic management functions
- Installation routines
- Command interface
- TME 10 Desktop (Graphical User Interface)

### 1.1.1 Operating System Independence

One of the goals of Tivoli is to provide management capabilities for systems, independent of which operating system they may be running. The operating system provides application programs with access to the hardware associated with a system. This hardware includes memory, disk storage, communication facilities, and so on. Though each operating system may provide different interfaces to the hardware it supports, the basic tasks that it carries out are common across all platforms. For instance, every operating system that supports disk storage devices must provide interfaces that an application can use to read and write data to those devices.

The TME 10 Framework encapsulates core operating system function in such a way that management applications may use standard interfaces to obtain information from and control the systems on which it operates. Currently, the TME 10 Framework is fully supported on various operating systems such as:

- Sun SunOS 4.1 and 4.2
- Sun Solaris 2.4 and 2.5
- HP HP-UX 9, 10.0, 10.1 and 10.2
- IBM AIX 3.2.5, 4.1 and 4.2
- Microsoft NT 3.51 and 4.0

In addition, the TME 10 Framework has been ported to run in other operating environments, including:

- Digital Unix 3.0, 3.2, and 4.0
- NCR SVR4.3 V2.02, 2.03 and 3.0
- Sequent V4.1.3, 4.2
- Solaris-ix86 V2.5 ad 2.51
- SGI-IRIX 5.3
- DGUX-88k 3.0 and 3.1
- Unixware 2.1.1
- Pyramid 1.1 and 5.4.2

The TME 10 Framework is an object-oriented environment in which operations are carried out through interfaces defined for various objects. The description just provided regarding the encapsulation of operating system functions implies that Tivoli has defined various objects which represent system resources. Management of these resources is accomplished by accessing these objects and invoking requests (methods) upon them.

Since the TME 10 Framework is, by definition, a framework to enable the management of multiple systems in a networked environment, there must be a way of invoking methods on objects that reside elsewhere in the network and independent of the operating environment controlling the resource. Architectures that provide such a capability have become commonly referred to as Object Request Brokers (ORBs). The ORB associated with the TME 10 Framework is discussed in the following section.

## **1.1.2 Distributed Object Framework**

In any distributed management environment there must be a communication mechanism that can be used between systems to handle the issuing of requests and the receiving of data. TME 10, which is based on an object-oriented design, uses the concept of an Object Request Broker (ORB) as its underlying communication mechanism.

An ORB is a mechanism that allows methods to be invoked on objects independent of their location. The best known and most widely implemented ORB architecture is that defined by the Object Management Group (OMG) and known as the Common Object Request Broker Architecture (CORBA). The Tivoli Management Framework includes an implementation of a CORBA compliant ORB.

### **1.1.2.1 A Short Digression on CORBA**

We do not intend this section to be a detailed dissertation on OMG's CORBA, but it is important to understand the basic concepts involved. The

OMG and their Object Management Architecture, the umbrella under which CORBA was developed, had its start in 1989 (coincidentally the same year that Tivoli was founded). The OMG is a consortium of well over 500 Information Systems vendors who understand the importance of the merger of object-oriented technologies and client-server computing. The members of OMG also understand the importance of standards that allow systems from various vendors to coexist and inter-operate.

CORBA is an industry standard designed to allow objects on various systems to communicate and inter-operate with each other. The concepts behind CORBA are really quite simple. At its core, CORBA allows implementations to completely hide the communications mechanism from the application developer. In addition, other aspects of CORBA provide services such as:

- An Implementation Repository (providing a directory service) to allow the current locations of objects to be dynamically determined and hidden from the application programmer.
- An Interface Repository to allow interfaces of newly introduced object types (or classes) to be queried dynamically so that existing applications can call methods on instances of these classes.
- Object Adapters to provide ORB specific services related to ensuring a method is invoked on the proper object using the appropriate interface.
- Interfaces to allow for the management of the ORB itself.
- Other facilities to handle security, transactions, persistence and so on.

For the CORBA purist, the above list will be considered incomplete, but be aware that CORBA is committed to addressing all of the important issues related to distributed objects. The CORBA architecture can be dissected in a variety of ways, but it is most frequently described as being composed of three levels of architecture:

- The ORB itself, which provides the communications infrastructure, as well as the tools required to compile IDL into bindings and stubs that allow applications to interface with the ORB in a standard way.
- A services level, providing support for transactions, persistence, security and many other requisite services.
- Common facilities providing towers of function related to common business application requirements such as a compound document architecture.

### 1.1.2.2 TME 10 Framework's ORB

After that short digression, the key point to be made is that the TME 10 Framework has, at its core, a CORBA compliant ORB that provides a common communication infrastructure across all platforms on which the TME 10 Framework operates. The operation of TME 10 applications consists of invoking methods on TME 10 Framework and TME 10 Application objects, that in turn communicate and pass data via the ORB.

The TME 10 Framework's ORB provides efficient transport for both small and large amounts of data. Therefore, the TME 10 Framework can be used for sending events, which typically consist of relatively small amounts of data (measured in kilobytes), or used for doing software distribution, which typically requires the transfer of large amounts of data (measured in megabytes).

Figure 1 shows how the ORB fits into a TME 10 environment.

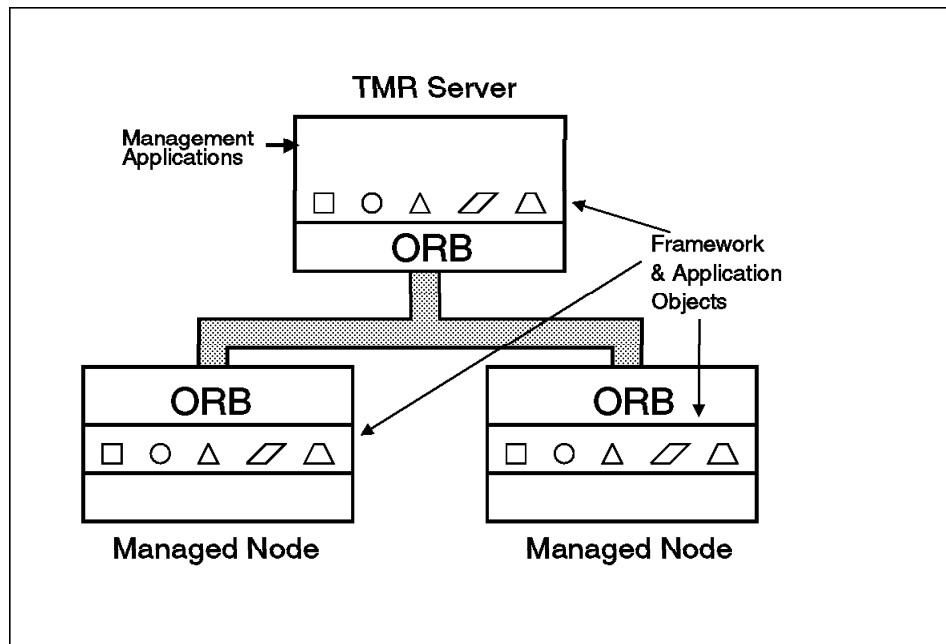


Figure 1. Role of the ORB in a TME Environment

As can be seen in this figure, the ORB provides the communications mechanism for the TME 10 environment. TME 10 applications consist of objects communicating with other objects. There are many object classes that are part of the TME 10 Framework. Instances of these classes provide basic functions such as transferring files and executing commands on remote systems. In addition, TME 10 applications may add their own object

classes to provide application-specific function. Either way, management operations are initiated through method invocations on these objects.

### 1.1.3 TME 10 Framework's Distributed Database

The primary purpose of a systems management framework is to gather and make available information about the resources that make up the distributed environment. The TME 10 Framework uses objects and an ORB as described above to gather the data as well as to provide interfaces for the management applications to access the data. The data we are discussing may represent information specific to a particular resource (for example, CPU utilization), or information about the managed environment (such as a list of all managed nodes). Some of this data may be relatively static in nature, while others (such as current CPU utilization) are dynamic by definition.

Access to all of the data associated with a managed environment is controlled through a set of interfaces which represent a distributed database. The underlying structure of this database and where it stores or accesses specific data is hidden from the user and applications. By providing an object-oriented interface to this distributed database and utilizing the framework's ORB to invoke requests upon the database, the framework will transparently access the required data independently of its physical location.

In fact, the database is spread over all of the systems that make up a Tivoli Management Region (TMR). TMRs are defined further in 1.2, "Tivoli Management Regions" on page 9. In each TMR, there is one system identified as the TMR Server and it provides the point of control for the distributed database. Much of the information about the TMR as a whole is stored on this system, but information about individual managed systems is typically stored on the managed system itself. Requests for data are typically handled first by accessing the TMR Server. If the data is controlled by another object in another system, the requester is provided information needed to locate and access that object and the request is completed by communicating directly with the controlling object. Of course, all of this is handled within the TME 10 Framework and the user and for the most part, the application developer, are unaware of these various mechanisms.

As mentioned above, the distributed database is represented by a set of interfaces or objects. Access to the information in this database is obtained by invoking methods on these objects. Therefore, we can modify the figure we previously showed to include the distributed database. The new figure is shown in Figure 2 on page 7.

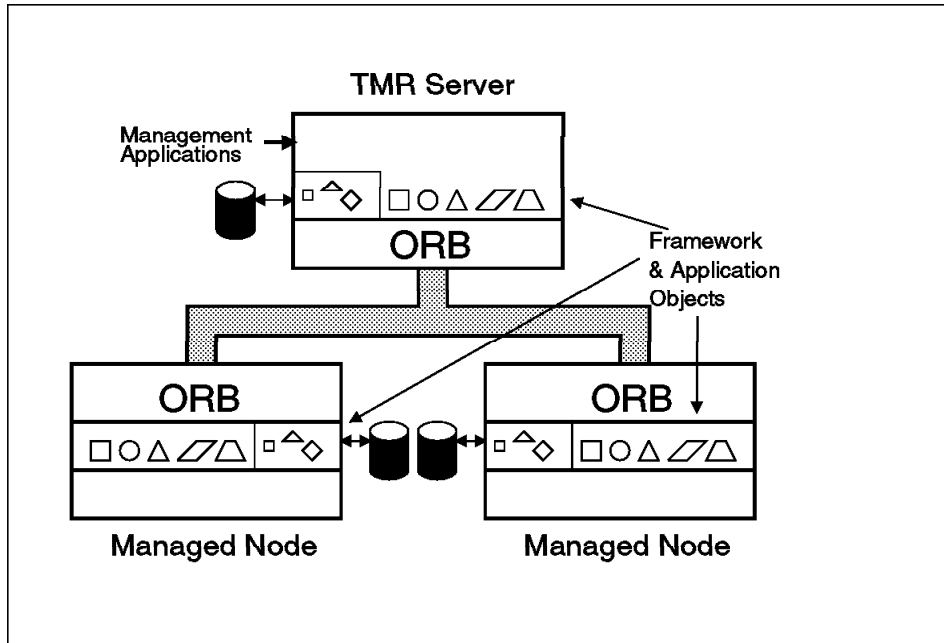


Figure 2. TME's Distributed Database

When methods are invoked to access the distributed database, the database objects will access the local files or memory representing the database and, if required, communicate to objects representing the databases on other nodes to perform the requested operation. Again, all of this is transparent to the user and programmer.

#### 1.1.4 Basic Management Functions

The TME 10 Framework is just that, a framework. In general, it primarily exists to provide enabling mechanisms for management applications. That is, the framework performs all of the necessary object resolution, communication and database access required so that the application developer can concentrate on providing management function rather than being bogged down with the details of cross-system communication and services.

If we think in basic terms about the requirement of a management framework, it must provide only a few core capabilities such as data transfer and remote command execution. In practice, of course, it must also provide security, transaction support and other services required of any robust application environment. All of these capabilities are built into the objects that make up the TME 10 Framework.

Applications simply build on these core facilities to provide capabilities such as software distribution, distributed monitoring and so on. For instance, one of the basic management functions provided by the TME 10 Framework is remote task execution. This function allows an administrator to execute a program on a remote system. The TME 10 Framework provides the facilities to store the program, or a shell script that invokes the program, in the distributed database and when the administrator requests it, to send the task to the target system and have it invoked.

Some of the core management facilities that are provided with the TME 10 Framework include:

- Installation routines (described in the next section)
- Capabilities to define and invoke tasks/jobs
- Capabilities to schedule jobs
- Obtaining rudimentary information about managed nodes
- Administer the TME 10 environment by performing operations such as:
  - Defining administrators and their roles
  - Creating new policy regions (groups of resources conforming to the same sets of rules or policies)
- Notification bulletin board to keep administrators aware of changes to the management environment.

### **1.1.5 Installation Routines**

For a distributed management environment to be successful, it must provide a mechanism to allow both its framework components as well as management applications to be easily installed and maintained across the systems being managed. The TME 10 Framework provides this capability. Once an initial system is installed, the framework provides simple mechanisms to push the required framework components, as well as management applications and maintenance out to other managed nodes. This capability requires little or no interaction on the target system itself.

These installation routines use a core mechanism built into the framework which is also utilized by the TME 10 Software Distribution application. TME 10 Software Distribution provides a more general purpose facility for distributing any application or data files to managed nodes.

### **1.1.6 Command Interface**

Once the TME 10 Framework is installed, there obviously must be interfaces for system administrators to request various functions to be performed. The TME 10 Framework provides both a graphical user interface (called the TME



Desktop and described in the next section), as well as a command line interface. The command line interface provides complete control of the managed environment and can easily be used by shell scripts to perform complex sequences of commands. When used in conjunction with applications such as TME 10 Distributed Monitoring, shell scripts can provide a high degree of automation by invoking various TME 10 Framework commands in response to the occurrence of specific events within the managed environment.

### **1.1.7 TME 10 Desktop (Graphical User Interface)**

The graphical user interface is a view of the TME 10 managed environment through the applied security filter for the current user. Desktops can be defined for individual administrators such that only the resources for which they have responsibility will be available to them. Information about each administrator's desktop is stored in the TME 10 database and, therefore, an administrator's desktop can follow him or her wherever he or she is authorized to log into the TME 10 environment.

---

## **1.2 Tivoli Management Regions**

Now that we have described the basic concepts behind the framework, we need to describe how these various facilities are used to manage your environment. A Tivoli Management Region (TMR) consists of one or more systems running the TME 10 Framework. In each TMR there is exactly one TMR Server. The TMR Server communicates with various managed systems via the ORB as described in the previous section.

In previous versions of the TME 10 Framework, there were two types of managed systems, Managed Nodes and PC Managed Nodes. It is important to understand the differences between these types of nodes.

A representation of a simple TMR is shown in Figure 3 on page 10.

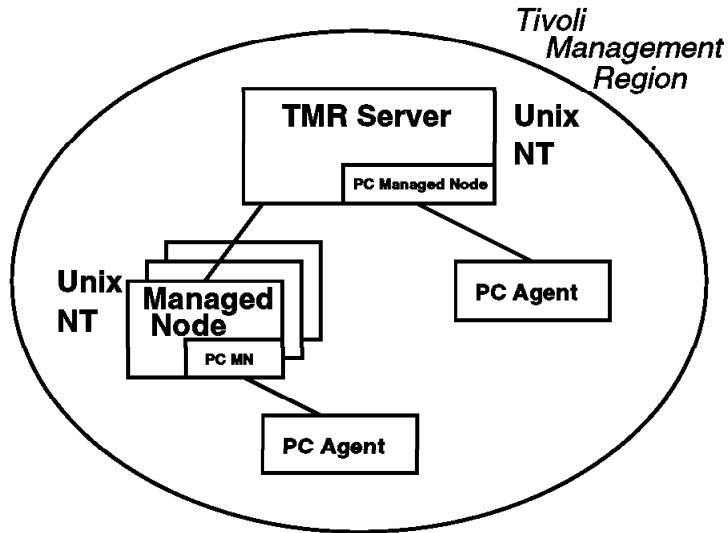


Figure 3. TME 3.1 Architecture

The **TMR Server** is responsible for maintaining the database and object registry. Recall that the database is distributed and spread among the various managed nodes in a TMR, but the TMR 10 Server provides the overall coordination of the distributed database.

The **Managed Node** is a client that also runs the TME 10 Framework, including the client portion of the distributed database managed by the TMR Server. This database contains objects and name registries related to the client and is linked to the server database via the Object Request Broker (ORB). Managed Nodes run complete versions of the TME 10 Framework and support any applications written to the standard TME 10 interfaces. In fact, the primary difference between the TME 10 Server and a Managed Node is that the server has overall control of the database. Otherwise, the full TME 10 Framework is installed and is functional on a Managed Node.

The **PC Managed Node** is an object in a Managed Node that represents a PC that is to be managed. The PC that is to be managed does not run the TME 10 Framework as described thus far. Instead, it contains and runs an agent that communicates with the PC Managed Node object on a Managed Node. The communication between the agent and the PC Managed Node object uses either TCP/IP or Netware's IPX protocol. Management applications interact with the PC Managed Node object, which then communicates with the agent running on the managed PC.

The PC Managed Node concept allows large numbers of PCs to be managed without requiring the full TME 10 Framework to be installed on each system. However, because the agent and the PC Managed Node do not communicate via the ORB mechanism, all of the functions that can be performed by a PC Managed node are included in the agent module. This set of functions is not extendable by the customer and provides support for only a small subset of TME 10 management applications.

### **1.2.1 Interconnecting TMRs**

Due to performance considerations mostly related to the distributed database (a database on every Managed Node), a TMR is limited to supporting a maximum of about 200 Managed Nodes in typical environments. There can be many more PC Managed Nodes, but due to the limited functions supported by PC Managed Nodes, this option is not always acceptable. However, for most companies a limit of 200 systems that can be fully managed is nowhere near adequate.

To allow for the management of many thousands of systems, the TME 10 Framework allows for the interconnection of TMRs. Each TMR still maintains its own database distributed between the TMR Server and the Managed Nodes that make up the TMR, but management applications can perform their functions across TMR boundaries transparent to the application. Once the TMRs are interconnected, the TMR Servers for each TMR exchange information about the managed objects in each TMR and the framework handles routing requests to the TMR Server in the appropriate TMR. The owning TMR Server then passes the request (method invocation) on to the managed object.

The design of a managed environment with regard to the number of TMRs, the number of systems in each TMR and how the TMR boundaries are defined (organizationally, geographically, and so on) is something that each customer must decide for themselves. Though there may be good reasons to create multiple TMRs in a single environment (such as for security considerations), there are many cases where multiple TMRs have been used simply due to the limited number of Managed Nodes that can be supported by a single TMR Server.

---

## **1.3 TME 10 Version 3.1 Limitations**

The architecture described above has some limitations in its current implementation. From a managed system standpoint, there are two options:

- Managed Nodes
- PC Managed Nodes

Managed Nodes have a complete implementation of the framework installed. This has benefits, but it also has associated costs. Specifically, the memory and disk space requirements for a managed node can exceed the requirements that you would sometimes like to impose on end-user systems. That is, for systems that will only be managed and never participate in managing resources within the network, you would like to minimize the resources (disk, memory) needed to enable the management of the system.

PC Managed Nodes provide a partial solution. The resource requirements for PC's that will be managed via PC Managed Nodes are much smaller than for a full Managed Node. However, since the mechanisms used to communicate with PC agents from PC Managed Nodes are not consistent with the rest of the framework and have limited function, PC's represented by PC Managed Nodes can not be managed by all Tivoli applications.

Therefore, the requirement is to provide TME 10 Framework support for a wide range of typical end-user systems that can take advantage of CORBA and the framework facilities, but does not require the resources demanded by today's Managed Nodes.

Extending the current framework to address these requirements resulted in the Lightweight Client Framework (LCF). LCF, which is the most significant of the many new features introduced by Version 3.2 of the TME 10 Framework, is the primary subject of this redbook.

LCF provides a solution that is an extension to the current architecture. With this extension, partial solutions such as the PC Managed Node will no longer be required. Like PC Managed Nodes, LCF requires only a small footprint on the managed system. Unlike PC Managed Nodes, LCF allows applications to use the power of the TME 10 architecture on all managed systems including lightweight clients. But we are getting ahead of ourselves. LCF will be described in the chapters that follow.

---

## 1.4 Summary

In this chapter we have provided a very brief description of the classic TME 10 Framework and outlined the requirements that led to extending the current architecture to include the Lightweight Client Framework. In the next chapter we provide an overview of the new features of Version 3.2 of the TME 10 Framework, including LCF.

---

## Chapter 2. TME 10 Framework V3.2 - What's New?

This chapter introduces the new features provided by Version 3.2 of the TME 10 Framework. These include:

- Lightweight Client Framework (LCF)
- Changes to the TME 10 Graphical User Interface (GUI)
- New commands for the TME 10 Command Interface
- RDBMS Interface Module (RIM)
- DynaText Enabling
- Web Interface

---

### 2.1 Lightweight Client Framework

The most significant new feature of Version 3.2 of the TME 10 Framework is the Lightweight Client Framework (LCF). LCF is an extension to the classic TME 10 Framework that increases scalability of TMRs while reducing the hardware and software requirements on managed systems.

#### 2.1.1 LCF Basics

LCF introduces three new object types that represent system roles in a TMR:

- Endpoint
- Endpoint Gateway
- Endpoint Manager

Though each of the above logically represents a different system's role in the TME 10 environment, it should be noted that a single physical system can contain more than one of the above object types.

##### 2.1.1.1 Endpoints

Endpoints are typically installed on systems that are considered 'managed only' 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, you will not typically be running the TME 10 Desktop or running TME 10 commands from an LCF Endpoint to manage other resources in the network.

The Endpoint function resides in the node to be managed. It runs as a small daemon, or background task. This daemon is called the spawner and is also often referred to by the name of its executable module, `lcf.d`. This daemon is responsible for executing methods at the request of a managing

system. Its only connection to and knowledge of the rest of the TME world is through an Endpoint Gateway.

When an LCF Endpoint is installed, a minimal number of files are installed on the managed system. Functionally, the only thing installed is the spawner 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 spawner. Methods that are downloaded to an Endpoint are cached at the Endpoint. As long as that method stays in the cache, it need not be downloaded again upon a second invocation of the same method. The cache on the Endpoint is a disk cache, and therefore is persistent across IPLs of the managed system.

#### **2.1.1.2 Endpoint Gateways**

An Endpoint Gateway is software that runs on a full 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 will depend on factors such as the available system resources, the number of Endpoints, network topology and so on. Currently one TMR Server can handle up to 200 Endpoint Gateways. One Endpoint Gateway can handle a thousand or more Endpoints.

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 Endpoints come online, they will either attempt to login to a specific Endpoint Gateway or broadcast a message searching for an Endpoint Gateway. Endpoint Gateways will receive these transmissions, and if responsible for the given Endpoint, will proceed with the login process. If the Endpoint is not in the Endpoint Gateway's list, the Endpoint Gateway will forward the request to the Endpoint Manager so that an Endpoint Gateway can be assigned to the Endpoint. The Endpoint Gateway's list of Endpoints for which it is responsible is stored and maintained by the Endpoint Manager described in section 2.1.1.3, "Endpoint Managers" on page 15.

- Listens for 'downcall' method requests from other nodes that are targeted for one of the Endpoints it is responsible for, and acts as a gateway for method invocations.

All operations to or from an Endpoint pass through an Endpoint Gateway. For downcalls, the Gateway is transparent. When it receives a method invocation targeted for an Endpoint for which it is a 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 an Endpoint needs to invoke an operation on another system, it must invoke a method on its own Endpoint Gateway. The Endpoint Gateway portion of the appropriate application 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.

Endpoint Gateways are automatically defined as MDist Repeaters for all of the Endpoints they serve. (See below for a description of MDist Repeaters). This gives you the benefit of an intelligent distribution mechanism with little or no administrative overhead.

***MDist Repeaters:*** Please see the *TME 10 Framework Planning and Installation Guide* for detailed information on MDist Repeaters. Briefly, they provide a fan out facility for the distribution of files and data in a TME 10 environment. Therefore, if the same file is being distributed to a set of LCF Endpoints using the same Endpoint Gateway, the file need only be sent once to the Endpoint Gateway and the Endpoint Gateway will then handle distributing the file to the individual Endpoints.

This can be a valuable feature in many environments. For example, Endpoint Gateways may be installed in various branch offices that are connected to a main site via a relatively slow link, The distributed data may only need to pass once along the slower link, and can then be distributed to the individual nodes on the local network at higher speeds.

### **2.1.1.3 Endpoint Managers**

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

- The Endpoint Manager maintains an Endpoint list, which keeps track of every Endpoint in a TMR. This list tracks which Endpoint Gateway each Endpoints is associated with. Based on site-specific settings, the Endpoint Manager reassigns Endpoints if an 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 Endpoints. This includes:
  - The name of the Endpoint: a "user friendly" name for easy use in the Tivoli Name Registry

- The Endpoint's interpreter type: a string denoting the platform and operating system of the Endpoint (for example, OS/2 or NT)
  - A unique object dispatcher identifier (odnum): the system identifier for the Endpoint
  - The name of the Endpoint Gateway 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 about the Endpoint, which is included in the Endpoint's broadcast message when looking for a new Endpoint Gateway.

Additional information on these policies and the connection process for Endpoints is presented in 3.4, "Endpoint Connection Process" on page 34.

#### 2.1.1.4 LCF Structure Summary

The following picture represents the new LCF structure within a TMR.

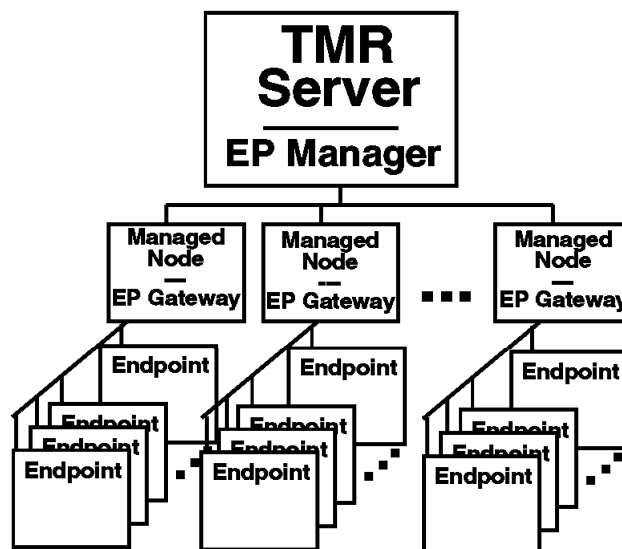


Figure 4. TMR Utilizing LCF

In the LCF environment, the TMR Server takes on the role of the Endpoint Manager. There will be some number of Endpoint Gateways, which currently must reside on traditional Managed Nodes. The number of Endpoint Gateways will be determined by a number of factors, but will



typically be much less than the number of full Managed Nodes in a pre-LCF environment.

The majority of the systems in a typical TMR will be LCF Endpoints. One Endpoint Gateway will be able to manage a thousand or more Endpoints.

### **2.1.2 How LCF Changes the TME 10 Topology**

Although we did not express this explicitly, the traditional TME 10 Framework provided a two-tier structure. That is, there were Servers and Managed Nodes (which may have also included PC Managed Nodes). One might consider PCs running PC Agents to be a third tier, but the framework architecture really stopped at the PC Managed Node object in the Managed Node. The PC Managed Node then communicated to the PC agent through a proprietary mechanism outside of the framework architecture.

LCF provides a true three-tier topology. In its current implementation, Endpoint Managers must reside on TMR Servers and Endpoint Gateways must reside on Managed Nodes. LCF Endpoints will make up the vast majority of systems in a TMR. These systems will have minimal footprint requirements.

The role of the Endpoint Gateway in the LCF architecture allows it to off-load much of the function that was previously required to be executed on a TMR Server. Though the limitation of a TMR Server to only manage a maximum of about 200 Managed Nodes still exists, the introduction of the LCF Endpoint Gateway and Endpoint allows an Endpoint Gateway to support thousands of Endpoints. The fact that the LCF architecture allows Endpoint Gateways to off-load some of the processing from TMR 10 Servers, implies that a single TMR can now contain many thousand Endpoints. Unlike the PC Managed Nodes that allowed many thousand systems to participate in a TMR in previous versions of the TME 10 Framework, LCF Endpoints are full participants and will therefore be fully supported by all LCF-enabled applications.

### **2.1.3 LCF and the TME 10 Desktop**

In general, during day-to-day activity, the existence of LCF is mostly hidden from the administrator. That is, TME 10 applications will fully support LCF Endpoints, and you will use Endpoints as subscribers to profile-based applications just as you used Managed Nodes in previous versions of the TME 10 Framework.

However, there are some additions and changes to the TME 10 Desktop that the administrator will notice. For instance, a new icon is added to the TME 10 Desktop to represent the Endpoint Manager. You can create, edit and

delete Endpoint Gateways 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.

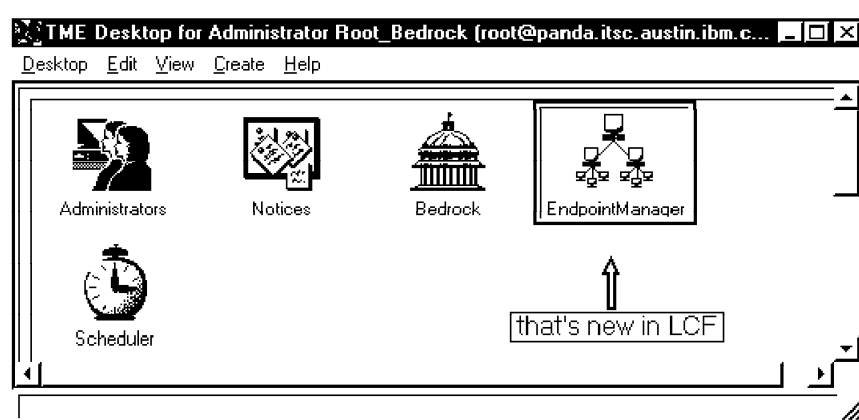


Figure 5. New Endpoint Manager Icon on TME Desktop

In addition, in the traditional TME 10 environment, Managed Nodes were displayed as such in the various policy regions. Due to the expected large number of Endpoints, and the screen area required to display potentially thousands of icons representing Endpoints, Endpoints are not displayed by default in the policy regions. However, they will be displayed as subscribers for any profiles to which they are subscribed.

#### 2.1.4 LCF and the Command Line Interface

Version 3.2 of the TME 10 Framework includes new commands specifically related to LCF. This section summarizes these new commands. Please refer to the *TME 10 Framework Reference Manual* and *TME 10 Framework Release Notes* for detailed information about these commands.

- |                 |  |
|-----------------|--|
| <b>winstlcf</b> | Installs an endpoint on a Unix workstation. For other platforms, such as Windows NT, alternate methods are used to install an Endpoint. For more information on installing LCF Endpoints, please see Chapter 4, "LCF Installation" on page 41.   |
| <b>wsetpm</b>   | Enables/disables a profile manager to operate in dataless mode. Since LCF Endpoints do not include a TME 10 database, profile information is not stored on the managed system the way it is for full Managed Nodes. Profile managers must be enabled for dataless operation to allow LCF Endpoints to subscribe to the profiles. |

<b>lcf</b>	Starts the spawner daemon on an Endpoint and installs or removes the daemon as a service on Windows NT.
<b>lcf.sh</b>	Starts the Endpoint daemon (lcf) on Unix endpoints.
<b>wcrtgate</b>	Creates an Endpoint Gateway
<b>wdelgate</b>	Deletes an Endpoint Gateway
<b>wdelep</b>	Deletes an Endpoint from the TME 10 database.
<b>wgateway</b>	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.
<b>wep</b>	Performs actions on the Endpoint List maintained by the Endpoint Manager. This command can list or alter information related to Endpoints.
<b>wadminep</b>	Performs a variety of administrative actions LCF Endpoints. In general, once Endpoints are installed, there is little that needs to be done to administer the LCF daemon. However, this command will be useful when first installing and testing LCF.
<b>wgeteppol</b>	Lists the body and the constant values of an Endpoint policy method. Use this command to extract a current policy method, which you might modify and then replace with the wputeppol command.
<b>wputeppol</b>	Replaces the body of an Endpoint policy method.

We will be showing examples of using most of these commands throughout the rest of this redbook.

### 2.1.5 LCF Summary

LCF adds a whole new dimension to the TME 10 Framework. It greatly enhances a TMR's scalability while drastically reducing the resource requirements placed on managed systems.

TME 10 applications will start taking advantage of LCF to provide robust function, while reducing the costs associated with maintaining the modules that will run on managed systems.

LCF will continue to be enhanced over time and one can certainly foresee LCF Endpoints being implemented on virtually all platforms in the not too distant future. This will lead to the capability of TME 10 applications to truly manage anything, anywhere!

---

## 2.2 RDBMS Interface Module - RIM

The RDBMS Interface Module (RIM) provides a generic architectural module within the TME 10 Framework for Tivoli applications using commercial relational database management systems (RDBMS).

RIM was introduced in V3.1 of the TME 10 Framework. However, in V3.2 it supports additional RDBMSs. RIM is shipped as part of the framework, although the framework itself does not utilize relational databases. Several TME 10 applications, such as TME 10 Inventory and Enterprise Console do require a relational database. Rather than ship RIM with the individual applications, it makes sense to ship this component with the framework so that any current or future applications that require a relational database can be ensured that the RIM interface is available.

RIM is a *TEIDL* (Tivoli Extended Interface Definition Language) interface that acts as an API for application developers using relational databases. Therefore, applications can be written independently of the relational database that may be installed in a particular environment.

The RDBMS systems currently supported by RIM are:

- Sybase 10.x
- Sybase 11.x
- Oracle 7.x
- DB2/6000
- Microsoft SQL 6.5

Support for additional database products may be added in the future.

The components that make up RIM include:

- RDBMS Vendor Independence Layer - A standard set of SQL interfaces that enables Tivoli applications to be open and RDBMS vendor independent.
- Data Model Independence - RIM provides a mechanism to bridge the 'impedance mismatch' between application data types and relational tables, which have contradictory design goals. Application data types need to be suitable for holding data needed for a variety of application specific requirements, where as relational tables are designed to be normalized for optimal storage, efficient queries and serving possibly multiple applications.
- Extensible API - RIM provides an extensible API to application developers for flexibility at the level that database access is desired; individual rows and columns level, or at a more abstract level.

- **Aggregate Types** - RIM supports application specific data aggregates which reflect the 'task-at-hand' approach, where you have all data elements contained in a single composite unit, without worrying about how the elements map into individual rows and columns in database tables.
- **Minimal Remote Object Calls** - RIM is modeled as a database access architecture that does not require excessive remote object calls. Limiting the number of remote calls to RIM objects provides improved application performance.
- **Scalable Database Architecture** - RIM presents enterprise databases as logical units to Tivoli applications. The physical location of the installations will not affect the applications, nor will the applications put any constraints on the physical database installations.
- **Transaction Model Support** - RIM supports TME transactions by allowing concurrent connections with conditional rollbacks.

### 2.2.1 Summary

RIM has been enhanced in V3.2 to support for additional databases. By implementing RIM, all TME 10 applications can utilize a single interface to relational databases, while allowing the customer to choose which RDBMS to install.

With RIM solidly in place, it is likely that we will see additional applications and maybe even some framework functions take advantage of relational databases. This can only enhance the capability to not only manage an environment, but to use information gathered through management applications to the benefit of the enterprise.

---

## 2.3 DynaText Document Viewer

With V3.2 of the TME 10 Framework, an online book browser called DynaText is provided. The DynaText browser is separately installable, but provides a powerful facility for viewing, searching and annotating the TME 10 documentation.

The DynaText browser is described in the *TME 10 Framework Release Notes* for the TME 10 Framework. The TME 10 Framework ships with online versions of the TME 10 Framework documentation as well as online books to assist you in using the features of the DynaText Browser.

New versions of the TME10 applications will include DynaText browsable documentation.

---

## 2.4 Web Interface

Version 3.2 of the TME 10 Framework includes integrated HTTP daemons that allow administrators to perform management functions through Web browsers. These daemons are automatically installed on TME 10 Servers, Managed Nodes and LCF Endpoints.

The HTTP daemon that executes on the TMR Server provides no default management capabilities. However, TME 10 applications can (and will) take advantage of this daemon to allow administrators to perform application specific function via their web browser.

The HTTP daemon that is part of the LCF daemon, allows administrators to query information specific to an LCF Endpoint. The use of the web interface to interact with LCF Endpoints is described in Chapter 6, “TME 10 Framework Web Interface” on page 75.

---

## 2.5 Summary

This chapter has provided an overview of the new major features of Version 3.2 of the TME 10 Framework. The rest of this redbook primarily addresses the Lightweight Client Framework, and how to plan, install, use and migrate to it.

---

## Chapter 3. Planning for TME 3.2

This chapter discusses issues related to planning for a TME 10 Framework installation. The topics address the traditional TME 10 Framework as well as LCF. We discuss:

- Hardware, software, and configuration requirements for Endpoint Managers, Endpoint Gateways, and Endpoints
- Network-related considerations
- Endpoint login process
- Other LCF-related considerations

---

### 3.1 Prerequisites

The TME 10 Framework software enables you to install and create several management objects that allow you to manage the resources in your network. You can install any or all of these objects depending on your organizational needs. Only the TMR Server must be installed. The following is a list of the management objects representing system roles provided by the TME 10 Framework:

- **TMR Server** - The TME 10 Server component includes the libraries, binaries, data files and graphical user interfaces needed to install and manage your TME 10 environment. Note that on UNIX platforms, the TME 10 Desktop GUI is installed with the TME 10 Framework, and on Windows NT the GUI must be installed separately.
- **Endpoint Manager** - The Endpoint Manager runs on the TMR Server. The Endpoint Manager maintains information related to known Endpoints and Endpoint Gateways. The Endpoint Manager is automatically installed on TME Servers.
- **Managed Node** - Managed Nodes run the same software that runs on a TMR Server. The primary difference is that the database it maintains only contains information about the objects controlled by that system. In addition, it does not include the Endpoint Manager function which is installed with the TMR Server.

System resource requirements are less than the TME Server because the database will only contain objects local to the Managed Node. The TME 10 Desktop GUI is automatically installed with the Managed Node on all UNIX-based systems. On Windows NT, the TME 10 Desktop is installed as a separate option. A Managed Node does not require the TME 10 Desktop if administrators will not perform TME 10 Desktop functions from that Managed Node. Note that TME 10 commands can be

issued from the node, allowing an administrator to perform most management functions from the Managed Node.

- **Endpoint Gateway** - The Endpoint Gateway provides the conduit through which management applications communicate with LCF Endpoints and Endpoints communicate with the rest of the systems in the TME 10 environment. A single Endpoint Gateway can support communications with thousands of Endpoints. In V3.2 of the TME 10 Framework, Endpoint Gateways must be installed on Managed Nodes.
- **Endpoint** - Endpoints are managed systems taking advantage of the Lightweight Client Framework. The TME 10 environment on these systems requires minimal resources, while providing the capability for these nodes to take full advantage of TME 10 applications written for LCF. The TME 10 Desktop is not installed with the Endpoint module. Since the Windows NT Desktop is not dependent on the TME 10 Framework being installed, it could be separately installed on NT Endpoints. In general, Endpoints should be systems not involved in the management of other systems.
- **PC Managed Node** - PC Managed Nodes are created on a proxy Managed Node, which provides communication between the PC and the TMR. However, the communication between the agent running on the PC and the PC Managed Node does not utilize the TME ORB. Instead, it is via a separate and proprietary communication mechanism that utilizes TCP/IP or IPX between the PC Managed Node and the limited function agent running on the PC.

Determining which systems in your network are best suited to provide these services requires that you consider a number of factors, such as:

- Required management capabilities
- Operating system
- Hardware requirements
- Server and client configuration
- Network topology and communications

### 3.1.1 Operating Systems

The following table identifies the operating systems on which TMR Servers and Clients can be installed. This table is intended as a quick reference, but the information within it might be changed frequently. Refer to *TME 10 Framework Release Notes* for the latest list of supported operating environments.



<i>Table 1. TME 10 Framework Operating System Support</i>				
<b>Operating System</b>	<b>TMR Server</b>	<b>Managed Node</b>	<b>Endpoints</b>	<b>PC Managed Nodes and PC Agent</b>
HP9000/700 HP-UX 9.01/9.05 Motif 1.2	Yes	Yes	Yes	-
HP9000/800+PA RISC 1.1 HP-UX 9.0 Motif 1.2	Yes	Yes	Yes	-
HP9000/700 or 800+PA RISC 1.1 HP-UX 10.01 or HP-UX 10.10 Motif 1.2	Yes	Yes	Yes	-
IBM RS/6000 & Power PC AIX 3.2.5, 4.1.2 through 4.1.5 or 4.2 Motif 1.2	Yes	Yes	Yes	-
Intel 486 or Pentium Windows NT 3.51 + SP1 or higher or Windows NT 4.0	Yes	Yes	Yes	Yes
Sun SPARC Solaris 2.3 through 2.5.1	Yes	Yes	Yes	-
Sun SPARC SunOS 4.1.2 or 4.1.3	Yes	Yes	Yes	-
OS/2 2.1, 3.0 and 4.0	-	-	Yes	-
OS/2 2.1, 3.0 with TCP/IP 2.0	-	-	-	Yes
Novel Netware 3.11, 4.1 and 4.11	-	-	Yes	Yes
Windows 95	-	-	Yes	Yes
Windows 3.1 and 3.11 with WINSOCK	-	-	Yes	Yes

### 3.1.2 Hardware Requirements

The following sections summarize the resource requirements for the various TME 10 resource types.

#### 3.1.2.1 TME Server Hardware Requirements

The type of system used as a TMR Server has significant impact on overall TME 10 performance. Note that only integer operations are used by the TME 10 Framework and applications. The TMR Server should provide good networking performance and should not be running a large number of non-TME 10 applications.

You should ensure that a TMR Server has enough swap space. Normally, this is two to three times the amount of system RAM. Also, you should

ensure that there are enough process slots available. The following table summarizes how you might check these resources in selected environments.

<i>Table 2. Swap Space &amp; Process Slots</i>		
<b>Operating System</b>	<b>Swap Space</b>	<b>Process Slots</b>
AIX	<code>/usr/bin/sar -v</code>	<code>/usr/bin/sar -v</code>
HP-UX	<code>/etc/swapinfo</code>	<code>/usr/bin/sam</code>
SunOS	<code>/etc/pstat -T</code>	<code>/etc/pstat -T</code>
SVR4	<code>/usr/bin/sar -r</code>	<code>/usr/bin/sar -v</code>
Windows NT	<b>Control Panel</b> - > <b>System</b> applet, select the <b>Performance</b> panel press the <b>Change</b> button	<b>N/A</b>

Increasing the amount of RAM in the TMR Server often improves performance. Determining the type and size of a system to use for the TMR Server depends on many factors, including the number of managed systems, and the mix of management applications and functions to be used. If you plan to have only tens of Managed Nodes, an entry level machine is acceptable, otherwise a higher class machine is recommended.

<i>Table 3 (Page 1 of 2). Disk Space Requirement Matrix.</i>					
<b>System</b>	<b>TMR Server (RAM/ Swap)</b>	<b>Managed Node (RAM/ Swap)</b>	<b>Endpoints (RAM/ Swap)</b>	<b>PC Managed Nodes (RAM/ Swap)</b>	<b>PC Agent (RAM/ Swap)</b>
Data General AViiON Series 530	64/128	48/96	-	-	-
HP9000/735	64/128	48/96	-	-	-
IBM RS/6000	64/128	48/96	-	-	-
Intel 486 or Pentium UnixWare 2.0	48/96	32/64	-	-	-
Motorola 88000 Series	64/128	48/96	-	-	-
NCR 3400/3500 Series	64/128	48/96	-	-	-
SPARC 10/SPARC 20 SunOS 4.1.x	48/96	48/96	-	-	-
SPARC 10/SPARC 20 Solaris 2.3-2.5.1	64/128	48/96	-	-	-
Intel 486 or Pentium Windows NT 3.51 or 4.0	48/128	32/64	16/-	16/-	16/-

*Table 3 (Page 2 of 2). Disk Space Requirement Matrix.*

<b>System</b>	<b>TMR Server (RAM/ Swap)</b>	<b>Managed Node (RAM/ Swap)</b>	<b>Endpoints (RAM/ Swap)</b>	<b>PC Managed Nodes (RAM/ Swap)</b>	<b>PC Agent (RAM/ Swap)</b>
NetWare	-	-	8/-	8/-	8/-
OS/2	-	-	8/-	8/-	8/-
Windows 3.x	-	-	2/-	2/-	2/-
Windows 95	-	-	8/-	8/-	8/-

### 3.1.2.2 TME 10 Managed Node

A managed node running the TME 10 Desktop should be a moderately powerful machine capable of running large win32-based or Motif applications. Of course, it depends upon the number of windows open at one time. TME 10 imposes no significant RAM or swap space requirements on a Managed Node except when actual operations are being performed. If you decide not to run the TME 10 Desktop from a Managed Node, you may reduce the RAM and swap space requirements depending on the model and the number of applications running.

Generally, the database requires a minimum of 10-15MB of disk space. See the *TME 10 Framework Release Notes* for more information.

### 3.1.2.3 TME 10 Endpoint Gateway

An Endpoint Gateway is software that runs on a full Managed Node, enabling the Managed Node to operate as a gateway between a cluster of Endpoints and the rest of the TME. The minimum hardware requirements are similar to a TME 10 Managed Node. Since Endpoint methods will be stored on the Endpoint Gateway (so they may be downloaded to the Endpoints cache as required, there will be some amount of additional disk space required. How much space is required will depend on the application mix in use. Depending on the number of Endpoints some additional memory may also be desired. Though designed to use resources efficiently, the exact requirements placed on an Endpoint Gateway will only be determined once applications are in place.

However, if the Managed Node will be performing as a Gateway for a large number of Endpoints, you should consider dedicating the system to its Endpoint Gateway function. This is not only for performance reasons, which would depend heavily on the kind of management operations you are performing, but also for reliability reasons. You would not want to lose an Endpoint Gateway because another application has rendered the system inoperable.

#### **3.1.2.4 TME 10 Endpoint**

An Endpoint requires only a small executable (the spawner daemon) to be initially installed and running. It maintains a cache of TME 10 methods that are downloaded and run on the Endpoint as needed. The default maximum size of this LRU method cache is 20.5MB. Most methods on the Endpoint will not put any significant resource requirements on the Endpoint system. However, since most Endpoint methods are application specific, you should refer to the documentation for the management applications you will be using for any specific prerequisites.

#### **3.1.2.5 TME 10 PC Managed Nodes**

The basic requirement is that it be a TME 10 supported system and that it has sufficient disk space. Creating the database object on a Managed Node has negligible impact on the disk space requirement of the Managed Node.

### **3.1.3 Server and Client Configuration**

This section describes the configuration requirements to support Endpoint Managers and Endpoint Gateways on various platforms.

#### **3.1.3.1 Endpoint Managers**

The TME 10 Framework can be installed on a wide range of UNIX servers and Windows NT machines. The CD image contains the binaries and libraries for all supported platforms.

The TME 10 Framework can be installed on UNIX machines running AIX 3.2.5, AIX 4.x, HP UX 9 and 10, Sun Solaris, and Sun OS, to name a few. It can also be installed on Windows NT 3.51 and 4.0.

For the framework, you'll require a minimum of 50 MB for the server database and at least 100 MB for binaries, libraries, man pages, and message catalogs. The total disk space requirement will depend on the TME 10 applications you will install and the size of your TME 10 environment.

Refer to the *TME 10 Framework Release Notes* and the application documentation for more detailed information on these requirements.

To install TME 10 Servers or Managed Nodes on UNIX and NT systems, you will need the root ID and password (for UNIX) and the Administrator ID and password (for NT).

During installation, the `tmeservd` id is added to the target system for some platforms (for example, HP-UX and NT). It is normally handled properly by the installation process, but if a user alters the `tmeservd` ID after the installation, this could result in unpredictable errors.

On other UNIX-based systems, the nobody ID must be configured.

To verify that the tmeservd and nobody IDs are properly configured on UNIX, verify that /etc/passwd contains information as shown here:

```
HP-UX      tmeservd:*:59999:59999: Reserved for TMR:/:/bin/false
AIX  nobody:*:4294967294:4294967294:./:
Solaris    nobody:*:60001:60001:./:
SunOS      nobody:*:65534:65534:./:
OSF/1      nobody:*:65534:65534:./:
```

**Read This!**

For SunOS clients, ensure that the default shell for root is /bin/sh. By default, SunOS has csh as the start-up shell. During install, TME 10 sends remote Bourne shell commands to the clients. If the default shell is csh, then these commands will fail.

### 3.1.3.2 Endpoint Gateways

When installing Managed Nodes on UNIX and NT:

- Ensure that you have the proper TME authorization, that is, you have at least install\_client and senior authority.
- It is essential for TME 10 Managed Nodes and Servers to be able to resolve each other's hostnames and IP addresses. All clients must be able to talk to the TME 10 Server as well as to other Managed Nodes via their hostnames and they must be able to map an IP address to the proper hostname. Using commands such as ping and nslookup you should verify that name resolution (resolving a name to an IP address) and reverse name resolution (resolving an IP address to a name) is working properly within your environment.

For example, if a TMR Server is named **dino** and the Managed Node is called **fred**, you might issue the following commands on the TMR Server (**dino**):

```
# nslookup fred
Server:  itsorusi.itsc.austin.ibm.com
Address: 9.3.1.74
```

```
Name:    fred.itsc.austin.ibm.com
Address: 9.3.1.47
```

```
# ping fred
PING fred.itsc.austin.ibm.com: (9.3.1.47): 56 data bytes
64 bytes from 9.3.1.47: icmp_seq=0 ttl=255 time=1 ms
```

```
64 bytes from 9.3.1.47: icmp_seq=1 ttl=255 time=1 ms
64 bytes from 9.3.1.47: icmp_seq=2 ttl=255 time=1 ms
```

Likewise, on the Managed Node (**fred**), the following commands would be issued:

```
C:\> nslookup dino
Server: itsorusi.itsc.austin.ibm.com
Address: 9.3.1.74
```

```
Name:      dino.itsc.austin.ibm.com
Address: 9.3.2.34
```

```
C:\>ping dino
PING dino.itsc.austin.ibm.com: (9.3.2.34): 56 data bytes
64 bytes from 9.3.2.34: icmp_seq=0 ttl=255 time=1 ms
64 bytes from 9.3.2.34: icmp_seq=1 ttl=255 time=1 ms
64 bytes from 9.3.2.34: icmp_seq=2 ttl=255 time=1 ms
```

### **Windows NT Client Considerations**

- Ensure that you have Administrator access on the target node.
- The `tmersrvd` user account and the `Tivoli_Admin_Priveleges` group account are automatically created during the installation process. If for whatever reason, this account needs to be created (or recreated) manually, the account must have `Log On Locally` privilege and its password must never expire.
- The Tivoli Remote Execution Service (TRIP) must be running on the target NT machine. TRIP provides a Remote Execution service for remote operations on Windows NT clients (basically equivalent to the REXEC service on UNIX). This service is used when installing an NT Managed Node.

To verify that TRIP is running, open the Control Panel, click on Services, and look for Tivoli Remote Execution Service, and ensure that it is started. You can also test this remotely by attempting to use the `rexec` command to execute a command on the target system.

TRIP must be installed manually on at least one NT system within your TMR. The install process uses the NT InstallShield mechanism. However, once it has been installed on one NT system, other NT systems within the NT domain may have it automatically installed transparently. This automatic installation is driven by the TME 10 installation process. Please refer to the *TME 10 Framework User's Guide* for more details on installing TRIP.

- To support the `odadmin start` command, TRIP monitors port 512. TRIP cannot operate if another exec service is running on port 512. You must shutdown any other exec services that are running on this port.
- The keyboard on the NT target machine should be set to US/English. If you are using a non-US/English keyboard, you can solve the problem by putting the `KBDUS.DLL` file in `WINNT\system32`. This file is the dynamic link library enabling the US/English keyboard.

---

## 3.2 Network Requirements and Considerations

Some of the key considerations related to network capabilities and configuration are described in the following sections.

### 3.2.1 Line Throughput

The minimal requirement is for a bidirectional, interactive TCP/IP line. In general if you have NFS ability between two points on your network, you should have no problem running TME 10 CLI commands and performing operations across slow lines. The installation of TME 10 clients across lines slower than 14.4KB is not supported. If you wish to install TME 10 clients that are accessible through slow lines, you should create a local TMR and perform the installation locally.

### 3.2.2 DNS (Domain Name System)

Although the network interfaces on a host and, therefore, the host itself are known by IP addresses, humans work best using the *name* of a host. In the TCP/IP world the *Domain Name System* (DNS) is a distributed database that provides the mapping between IP addresses and hostnames.

Any application can call a standard library function to look up the IP address (or addresses) corresponding to a given host name. Similarly, a function is provided to do the reverse lookup - given an IP address, look up the corresponding hostname.

From a Tivoli Management Environment perspective, methods access the DNS through a *resolver*. On UNIX hosts the resolver is accessed primarily through two library functions, *gethostbyname* and *gethostbyaddr*. Both of these interfaces are used by the TME 10 Framework.

An application must convert a host name to an IP address before it can ask TCP to open a connection or send a datagram using UDP. The TCP/IP protocols within the kernel know nothing about the DNS.

We will mention several times that correct name resolution and reverse (address to name) resolution is critical to the proper operation of the TME 10 Framework.

Of course, DNS is not the only mechanism that can be used to resolve names and addresses. You can also use NIS as well as local `etc/hosts` files.

### 3.2.3 DHCP (Dynamic Host Configuration Protocol)

TME 10 provides support for networks that utilize dynamic IP addressing. This support is provided for the following managed resources:

- Windows NT Managed Nodes
- Windows 3.x, Windows 95 or Windows NT running a PC Agent
- All LCF Endpoint clients

DHCP support for Windows NT Managed Nodes requires that your TMR Server be running Windows NT. UNIX TMR Servers do not provide the support required for DHCP Managed Node clients. Other name resolution services such as DNS or NIS do not normally communicate with DHCP. When a TME 10 Managed Node connects with the TMR Server, the Managed Node passes its current IP address to the Server. If the IP address is different from the previous one for the Managed Node, the server updates its address mappings. The TMR Server can resolve IP address changes caused by DHCP as well as those caused by moving a machine to a new subnet. To enable IP address resolution on a TMR Server, you must use the following command:

```
oadamin allow_dynamic_ipaddr TRUE
```

The synchronization of a PC agent's IP address with that maintained by the PC Managed Node is handled by DHCP service, which is separately installed on the TMR Server. It receives the IP address and PC Managed Node names from the PC agent, then relays that information to the PC Managed Node.

### 3.2.4 DDNS (Dynamic DNS)

The dynamic domain name system is needed in correlation with DHCP. Because the IP address of an Endpoint may vary if you use Dynamic Host Configuration Protocol (DHCP), you must be certain that the DNS host name and IP resolution tables will be updated correctly. DDNS keeps track of the used IP addresses and the corresponding host names.

In AIX the DDNS update is handled by using special parameters in the DHCP configuration file (`/etc/dhcpsd.cnf`). The parameter `updatedDNS` specifies a



program that will be called every time DHCP assigns a new IP address. It has to update the DNS database and refresh the DNS daemon to enable the change. For example,

```
/usr/sbin/dhccpaction <hostname><domainname><ipaddr><leasetime>REC NIM
```

---

### 3.3 Tivoli Management Regions

To manage thousands of resources that are geographically dispersed, the TME 10 Framework enables you to logically partition your managed resources into a series of loosely coupled Tivoli Management Regions (TMRs). Each TMR has its own server and therefore controls its own TME database. When TMRs are interconnected, the associated TMR Servers exchange lists representing the managed objects within their respective TMRs. This allows management applications to see references to objects in both TMRs and invoke operations on them.

If a management application invokes a method on an object in another TMR, the local TMR Server will pass the method request to the TMR Server responsible for the managed object. The TMR Server in the target TMR will then handle passing the method invocation request onto the target object.

Several TMRs can be connected together to coordinate activities across the network, enabling large-scale systems management and offering the ability for remote site management and operation. This capability was all-important in previous versions of the TME 10 Framework. With the introduction of LCF, the number of managed systems supported by a single TMR has dramatically increased. Therefore, this particular reason for supporting multiple interconnected TMRs may not apply in as many environments as it did previously.

However, there are still valid reasons for creating multiple TMRs and interconnecting them. Some of these reasons include organization structure, network topology, security requirements and so on.

Two TMRs can be connected through a:

- **One-way connection** - Only one of the TMRs has visibility to the managed objects in the other.

This scheme may be useful when a central site is responsible for administering several remote sites, but none of the remote sites have any need to manage resources at the central site. This scheme does not preclude having local administrators for the remote TMR. However, it does provide the capability for the central site to provide some level of management, if necessary, or to perform specific management functions that may be more global in scope. For example, the central site may be

responsible for the distribution of a new version of a corporate wide application.

- **Two-way connection** - This is the more typical connection type, as was described in the introductory paragraph above. Administrators and applications in each TMR have visibility and the ability (within policy) to manage resources in the other.

---

### 3.4 Endpoint Connection Process

The default mechanism that Endpoints use during their initial login phase is a broadcast sequence for finding a valid Endpoint Gateway. This broadcast is limited to the subnet of the Endpoint and if unsuccessful, will be repeated every five minutes by default. (The wait between broadcasts and the number of re-broadcasts can be configured.) Therefore, it is not very obtrusive to the network.

The following information on the login sequence for Endpoints will help you to understand where and how to configure initial Endpoint connections. Further details are presented in Chapter 5, “Configuring the LCF Environment” on page 61.

#### 3.4.1 How an Endpoint Connects

By default, when the spawner (lcf daemon) is installed, there is no configuration provided. When it is started for the first time, it must locate a gateway to which it can attach. To get connected to an Endpoint Gateway, the Endpoint initiates the sequence shown in following picture

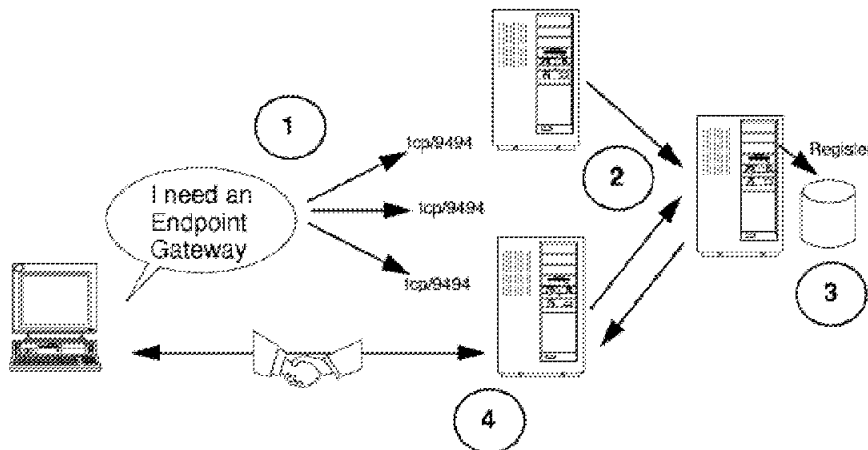


Figure 6. Initial Endpoint Connection

The numbered steps are as follows:

1. The Endpoint broadcasts on TCP port 9494, asking to be connected. By default, all of the Endpoint Gateways listen on this port, so that any active Endpoint Gateway in the same IP network will receive the broadcast request.
2. The Endpoint Gateways do not directly accept the Endpoint request, but instead forward it to the Endpoint Manager (which is also the TMR Server).
3. The Endpoint Manager registers the new Endpoint in the TME 10 database and assigns it to an Endpoint Gateway. A number of policy methods are provided to allow you to control how an Endpoint Gateway is chosen and execute any other automatic functions. For example, you may want to subscribe the new resource to profile managers, or alert an administrator that it has been connected.
4. The chosen Endpoint Gateway responds to the Endpoint connection request. The Gateway and Endpoint perform an initial handshake to establish their identities and generate encryption keys for authentication purposes.

### 3.4.2 Sequence of an Endpoint Connection

This section presents a slightly more detailed description of the Endpoint connection.

1. Endpoint starts the spawner (lcmd) looking for an Endpoint Gateway:
  - Broadcasts everywhere 255.255.255.255
  - Sends a UDP frame
  - Asks for socket port 9494 (by default)
2. An Endpoint Gateway on the same subnet:
  - Listens on port 9494 and receive the Endpoint's broadcast.
  - If the Endpoint is not already in its Endpoint list, forwards the broadcast frame to the Endpoint Manager.
3. The Endpoint Manager performs the following:
  - Checks if the Endpoint is a known node in TMR
  - Checks the Endpoint's characteristics against the following policies:
    - **allow\_install\_policy** - Controls which Endpoints are allowed to log into the TMR.
    - **select\_gateway\_policy** - Returns a list of Endpoint Gateways that may be used by this Endpoint to the Endpoint Manager. The

Endpoint Manager will walk this list until it finds an active Endpoint Gateway. That gateway will be notified that it should contact the Endpoint and complete the login process.

- **after\_install\_policy** - Performs a series of actions for the Endpoint after its first successful login. For instance, it may subscribe the Endpoint to a set of Profile Managers. This policy script is executed locally on Endpoint Manager
- **login\_policy** - A script that performs a series of actions for the Endpoint after every login. This policy script will be executed on the Endpoint Gateway. The Endpoint Manager takes care of distributing this policy script to the Endpoint Gateway as soon as it is identified.

Please refer to section 5.3, “Controlling an Endpoint Login Through Policies” on page 64 for more details.

- Updates the Endpoint Gateway List
  - Notifies the Endpoint Gateway that a new Endpoint has been assigned to it.
4. Endpoint Gateway sends back to the calling Endpoint a UDP frame, notifying it of its own IP address.
  5. Now the Endpoint can complete the login procedure:
    - The spawner writes the successful connection to the file `last.cfg`. This file will be overwritten each and every time the Endpoint successfully connects (logs in) to an Endpoint Gateway.
    - Writes additional login data to a binary file `lcf.dat`.
    - Listens for method calls on its socket port.
  6. Endpoint Gateway executes the **login\_policy** and starts listening for upcalls or downcalls from or to this Endpoint.

More details on the `lcf.dat` and `last.cfg` file are presented in Chapter 5, “Configuring the LCF Environment” on page 61.

### 3.4.3 Limiting Endpoint Broadcasting

As previously mentioned, Endpoint broadcasting does not adversely affect the network. The broadcast message is small, limited to a single subnet and will only occur every five minutes by default, until the login is successful or the maximum number of attempts (a configuration option) has been reached.

However, you can restrict the Endpoint broadcasting by creating a configuration file or passing parameters to the spawner using the `-D`

`lcs.login_interfaces` option of the `lcsd` command. This parameter allows you to specify a list of Endpoint Gateways that the Endpoint may use to login to the TMR. It will attempt to contact the Endpoint Gateways in this list directly, before performing a broadcast.

Gateways in the `lcs.login_interfaces` list are specified using the following format:

```
<gw_addr>+<gw_port>:<gw_addr>+<gw_port>
```

As an example, the following parameter could be specified:

```
-D lcs.login_interfaces=9.3.16.192+9494:9.3.16.255+9494
```

In the above example, we declare that the Endpoint's login Endpoint Gateway has an IP address of 9.3.16.192 and, in case the gateway does not respond after a specified `time_out`, `lcsd` will send a `subnet_directed` broadcast to 9.3.16.255 (all hosts on subnet 9.3.16.0).

Ensure you specify the correct network mask that will work in conjunction with the network mask used by your system.

Future versions of LCF may provide additional options for controlling broadcasts while still enabling dynamic discovery of appropriate gateways.

If broadcasting must be used, you can control which Endpoints might be assigned to a particular Endpoint Gateway through policies. For details and examples of implementing Endpoint's policies see 5.3, "Controlling an Endpoint Login Through Policies" on page 64.

---

### 3.5 General LCF Planning Considerations

At the time this redbook was written, LCF enabled applications were not available so we can not include specific recommendations based on actual experiences. Rather, we summarize in this section some of the general considerations that exist when planning an implementation of TME 10 utilizing LCF.

When the LCF enabled applications become generally available, (probably by the time you read this redbook), it is likely that the vast majority of systems in your network will be candidates for LCF Endpoints. Low-cost PCs or UNIX workstations are effective as Endpoints in an LCF environment because each is connected to a Managed Node, which runs a server-class framework capable of managing multiple simultaneous connections. Endpoints, with less computing power, can off-load some of their work to the Managed Node (Endpoint Gateway), which continues to perform the same

roles that it does currently, such as running the TME desktop, functioning as an Mdist Repeater, and acting as an ADE development seat, as well as providing the LCF Gateway functions.

This pairing of Endpoints and Managed Nodes enables the enterprise to scale to thousands of Endpoints for each Managed Node and ensures that the Endpoints are still fully able to be managed by the TME 10 applications.

In terms of scalability, LCF is an order of magnitude better than the existing server-class framework. There are two pieces to the increased scalability:

- The larger numbers of nodes each TMR can accommodate
- The computing load each can handle

In a Lightweight Client Framework environment, each TMR Server supports up to 200 Managed Nodes, but each Endpoint Gateway can support a far larger number of Endpoints. This dramatically increases the number of Managed Nodes in a 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. As the number of Endpoints and the number of operations for each Endpoint increase, the quality of performance will naturally decline. But, with the addition of an Endpoint Gateway, you can support a larger number of Endpoints without a performance decrease. It is important to determine the right balance between the number of Endpoint Gateways and Endpoints.

But numbers are only part of the equation in determining the quality of service. Another factor is your organization's model of network management. If, for example, your organization distributes software from a central site to, say, 5,000 machines connected to a single Endpoint Gateway, the distribution will proceed slowly, but will complete successfully. On the other hand, if 5,000 Endpoints connected to a single Endpoint Gateway each try to initiate a software pull, the quality of service will be unacceptable. In addition to considering the number and ratio of Endpoints and Endpoint Gateways, consider the system management applications your organization will use.

In summary, LCF Endpoints over-load the Endpoint Gateway delegating the storing and managing of Tivoli objects. After the login phase the necessary data is transferred to the Endpoint. As more applications are installed and

used, more data and methods will be downloaded to the Endpoints from their Endpoint Gateway.

Organizing and distributing Endpoint Gateways and Endpoints by

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

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

---

### 3.6 Summary

This chapter has provided an overview of information related to planning for a successful installation of one or more TMRs. More detailed information is presented in the *TME 10 Framework Planning and Installation Guide*. With the introduction of LCF, planning the mixture of Endpoints and Endpoint Gateways will be critical. Unfortunately, we can not provide much specific guidance in this area until the LCF-enabled applications become available.

The next chapter addresses the installation of TME 3.2-based systems. The primary differences introduced in the installation of the TME 10 Framework V3.2 relates to LCF. Therefore, that is the area we concentrate on when describing the installation processes.





---

## Chapter 4. LCF Installation

This chapter describes the installation of LCF Endpoint Managers, Endpoint Gateways and Endpoints. In V3.2 of the TME 10 Framework, Endpoint Managers must reside on TMR Servers, Endpoint Gateways must reside on full Managed Nodes and Endpoints can reside on any of the supported platform types with no other TME 10 prerequisites.

We provide examples of installing the various components through command line interfaces as well as the GUI interfaces that are available. For a more detailed description of the CLI and GUI commands, refer to the *TME 10 Framework Planning and Installation Guide* and *TME 10 Framework Reference Manual*.

---

### 4.1 Endpoint Managers

With the introduction of the Lightweight Client Framework technology, the TMR Server takes on a new role of Endpoint Manager. Though you can install a TMR based on Version 3.2 of the TME 10 Framework without utilizing LCF, the Endpoint Manager is installed when you install a TMR Server. The Endpoint Manager is required to create Endpoint Gateways which in turn are required to manage LCF Endpoints. Therefore, as with any TMR, the first thing we must do is install the TMR Server and Endpoint Gateway.

#### 4.1.1 Creating an Endpoint Manager

As just mentioned, an Endpoint Manager is created when a TMR Server is installed. We will not review the details of installing a TMR Server. This is well documented in the product documentation, specifically the *TME 10 Framework Planning and Installation Guide*. As a brief summary, the installation is accomplished through the `wserver` command on UNIX systems and through the `setup.exe` InstallShield program on Windows NT. On UNIX, you have the option of specifying all required information on the command line, or utilizing a GUI to prompt you through the options.

Note also, that the `wserver` command installs both the TME Server and the TME 10 Desktop. On NT systems, the TMR Server and TME 10 Desktop are installed via separate procedures.

Either way you install, once the installation is complete, the Endpoint Manager will be available. In Figure 7 on page 42, we show the TME Desktop after a new installation. If you are familiar with previous versions of the

TME 10 Framework, you will notice only one significant difference; the Endpoint Manager icon.

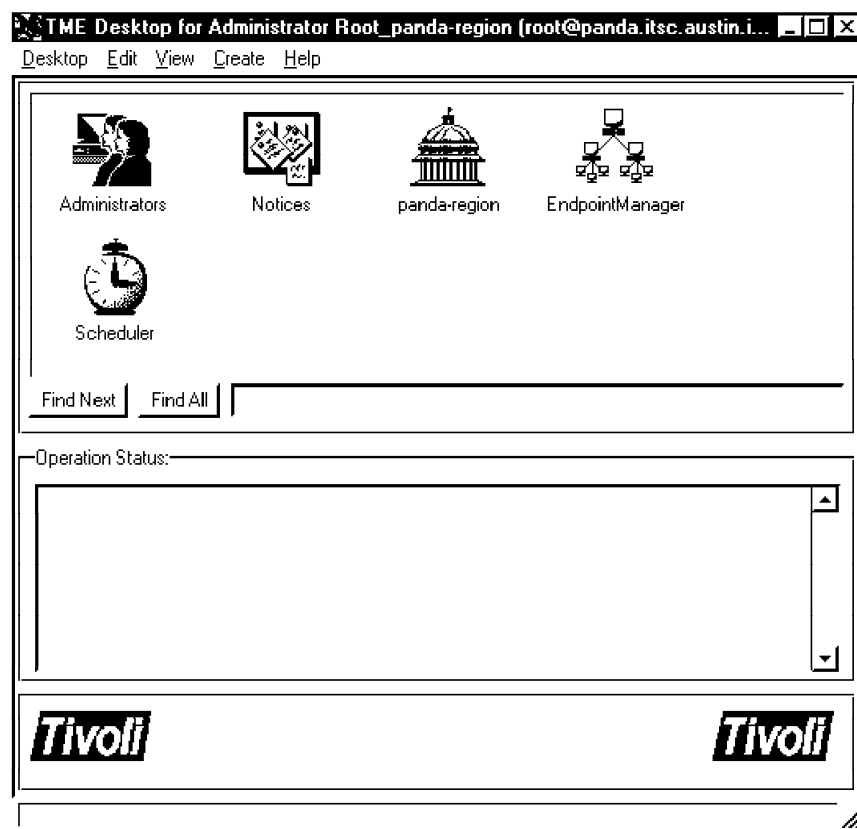


Figure 7. TME 10 Desktop with Endpoint Manager

From the command line interface, you can verify the Endpoint Manager exists through the `wls` command. An example of this command is shown in Figure 8

```
# wls
Notices
Administrators
panda-region
EndpointManager
Scheduler
```

Figure 8. Output of `wls` Showing Endpoint Manager

If using the TME 10 Desktop, we might try opening the Endpoint Manager. Figure 9 on page 43 shows the window that is displayed. At this point it is not very interesting since no Gateways or Endpoints exist.

In the next section, we discuss the creation of Endpoint Gateways.

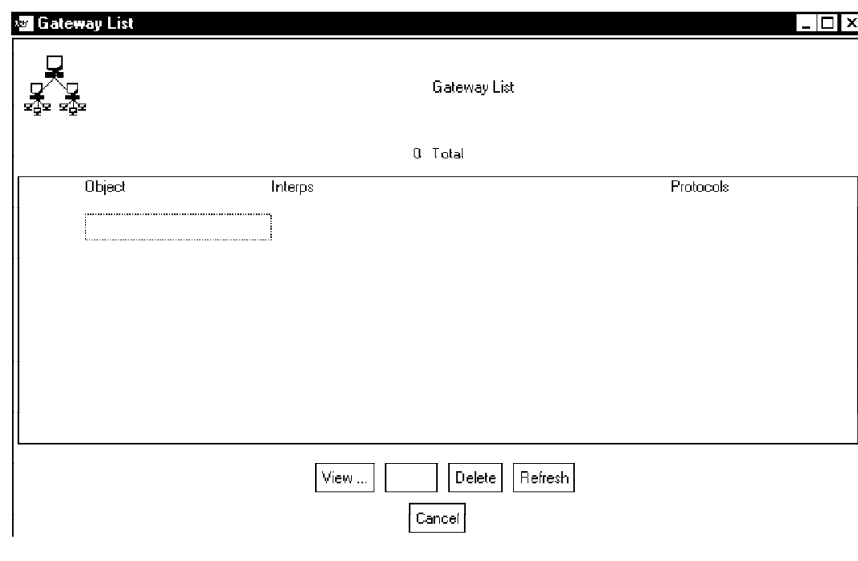


Figure 9. Endpoint Manager Immediately After Installation

## 4.2 Endpoint Gateways

Endpoint Gateways, as the name implies, provide the communication channel between an LCF Endpoint and the rest of the TMR. Aside from providing the communications conduit, the Endpoint Gateway is key in the scalability of TMRs, as it also off loads much of the function previously performed by TMR Servers.

In Version 3.2 of the TME 10 Framework, Endpoint Gateways must reside on TME 10 Managed Nodes. Therefore, the first step in creating an Endpoint Gateway is to create a Managed Node. Managed Nodes are created using the same procedure as in the pre-LCF environment.

Unlike, Endpoint Managers that are automatically created on any TMR Server that is installed, Endpoint Gateways must be explicitly created on a Managed Node after the Managed Node is created. We do not take you through the specifics of creating/installing a Managed Node. Refer to the *TME 10 Framework Planning and Installation Guide* for details.

Once a Managed Node is created via the TME 10 Desktop or command line interfaces, we can choose to install an Endpoint Gateway on that system. Don't forget that the TME Server is also a Managed Node. Therefore, the Endpoint Gateway can also be installed on the TMR Server.

### 4.2.1 Creating an Endpoint Gateway - Command Line Interface

The following steps may be taken to install an Endpoint Gateway via the command line interface.

1. Run the Tivoli setup environment script.

```
# . /etc/Tivoli/setup_env.sh
```

2. Backup the object database. By default, the backup is stored in \$DBDIR/../backups

```
# wbkupdb
```

3. Install a Gateway.

The command below creates an Endpoint Gateway called GW-goban on managed node goban. The -h parameter specifies the managed node and -n specifies the gateway name. By default, the Endpoint Gateway will communicate with Endpoints on port 9494. However, as we have done, you can use the -p portnumber parameter to use a different port. For more information on the wcrtgate command, refer to the TME 10 Framework Reference Manual.

```
# wcrtgate -h goban -n GW-goban -p 9495
```

4. Verify the new gateway is installed using the wgateway command as shown below:

```
# wgateway GW-goban describe
Object      : 1352741012.3.21#TMF_Gateway::Gateway#
Hostname    : goban.austin.ibm.com
Port       : 9495
Timeout    : 300
```

5. Backup the object database. By default, the backup is stored in \$DBDIR/../backups

```
# wbkupdb
```

### 4.2.2 Creating an Endpoint Gateway - GUI

The following steps can be used to create an Endpoint Gateway via the TME 10 Desktop.

1. Create a backup of the object database. Select **Desktop->Backup**. Select all the available nodes and click on **Start Backup**. Click on **Close**

when the backup is finished. By default, the backup is stored in \$DBDIR/./backups.

2. From the Endpoint Manager pop-up menu, select **Create Gateway...**

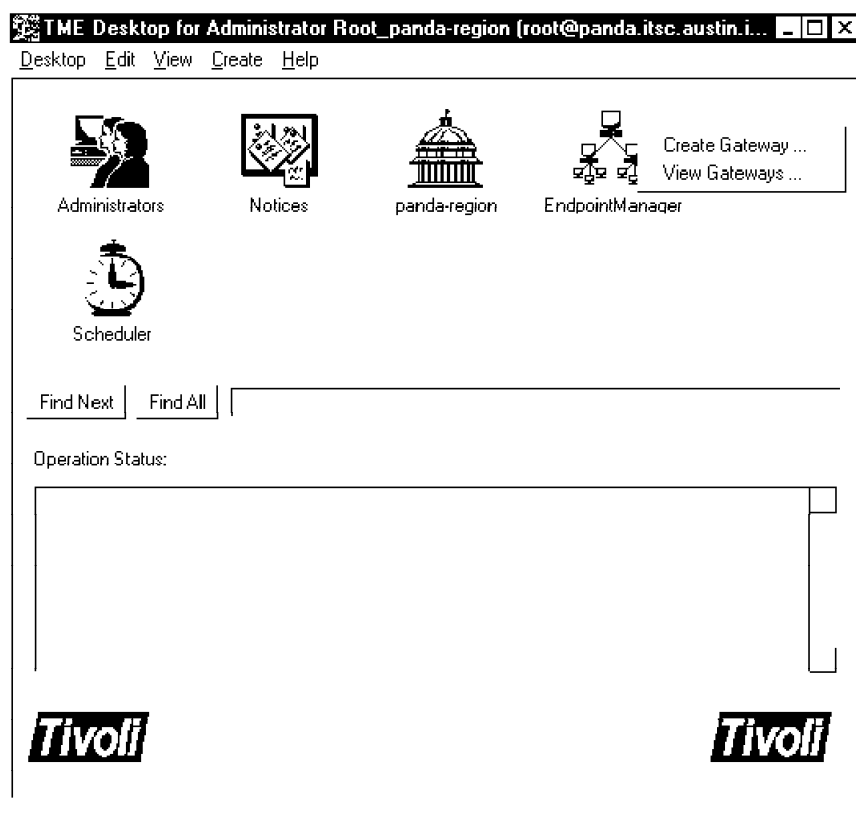


Figure 10. Gateway Installation

3. The Create Gateway dialog screen appears. The information to be filled into this dialog is the same as can be specified via the command line. Select **Create and Close** when finished. Note that we have chosen to use port 9495 for communication with our Endpoint Gateway rather than the default port of 9494. The reason for this was purely to provide an example of doing so. However, in a test environment, using an alternate port is helpful as you may have multiple test Endpoint Gateway's, and would like to easily limit which Endpoints connect to each without having to create Endpoint policy scripts. See section 5.3, "Controlling an Endpoint Login Through Policies" on page 64.

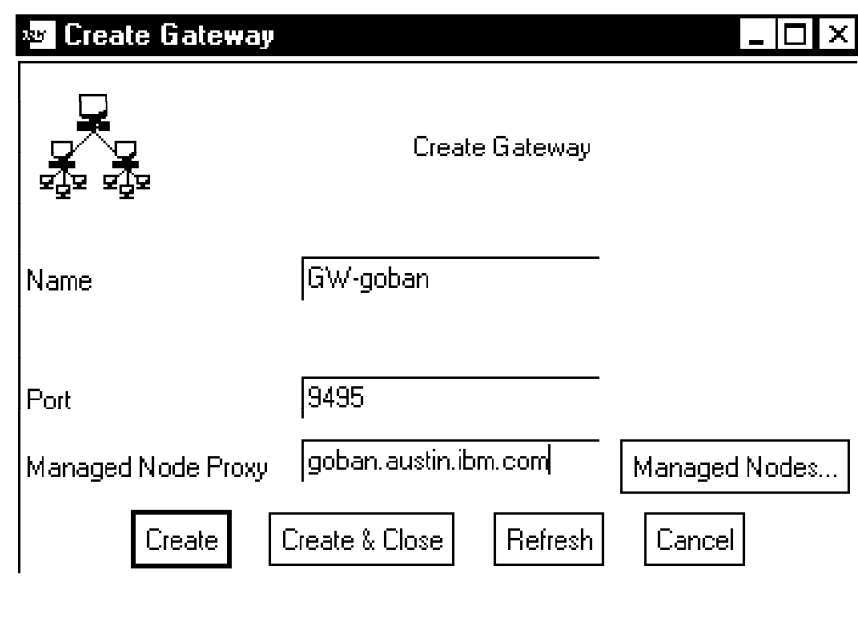


Figure 11. Create Gateway Dialog

4. Verify the gateway is successfully created. From the Tivoli Desktop, double click on Endpoint Manager, and you should see a panel similar to Figure 12 on page 47. In this figure you can see a list of currently defined Endpoint Gateways. You can select each Endpoint Gateway and use the **View** button to see a list of Endpoint associated with that Endpoint Gateway.

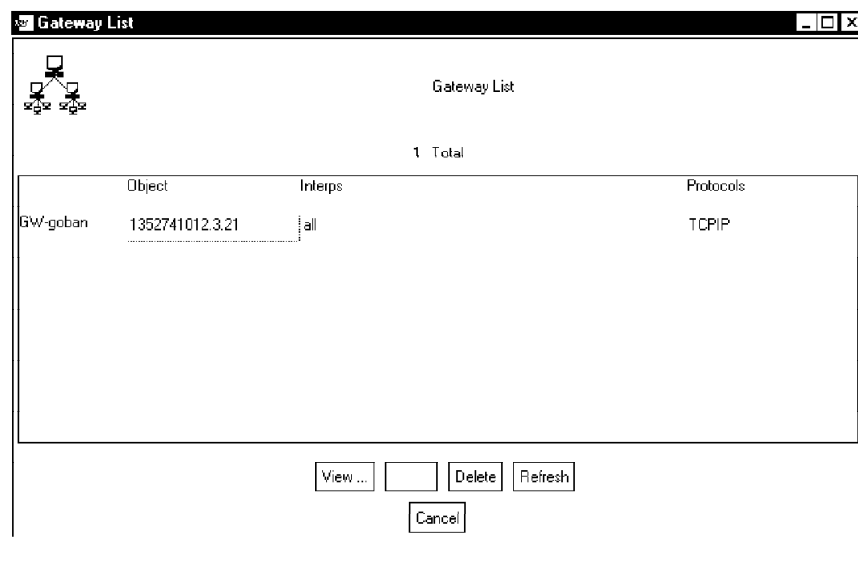


Figure 12. Gateway List

5. Create a backup of the object database. Select **Desktop->Backup**. Select all the available nodes and click on **Start Backup**. Click on **Close** when the backup is finished. By default, the backup is stored in `$DBDIR/./backups`.

### 4.3 Endpoints

Generally speaking, LCF Endpoints are machines not extensively used in your daily operations of managing a network. In other words, LCF Endpoints are typically end-user systems or even application servers that you want to manage while requiring a minimal footprint of TME 10 code.

Indeed, the LCF Endpoint binaries require just a few hundred kilobytes of disk space on the managed system. In addition to the Endpoint binaries, the LCF Endpoint will create a disk cache whose size is configurable, but defaults to 20.5 MB. This disk cache is used to store any methods that will be executed on the Endpoint by management applications.

The following platforms are supported as LCF Endpoints:

- AIX 3.2.5, 4.1, 4.2
- HP-UX 9.0, 9.1, 9.3, 9.5, 10.0, 10.1, and 10.2
- NetWare (server only) 3.12 and 4.1

- OS/2 Warp 3.0 (requires FixPack 27), Warp 4.0, and Warp Server 4.0 (also requires FixPack 27)
- Solaris 2.3, 2.4, and 2.5
- Sun/OS 4.1.2 and 4.1.3
- Windows NT 3.51 (with SP1) or 4.0 (includes SP1 and SP2). No support for multi-user variations of Windows NT.
- Windows 95
- Windows 3.x

For more details, refer to the *TME 10 Framework Release Notes*.

The installation procedure differs between the different platform types. For UNIX-based systems, the `winstlcf` command is used. For other systems, installation is through a platform specific install command, such as `setup.exe(InstallShield)` on Windows. However, even on windows, a GUI-based install is not required. Command line parameters can be used to specify all required information. The mass installation of a large number of LCF Endpoints could be accomplished via login scripts, TME Software Distribution File Packages (for nodes that are currently a Managed Node or PC Managed Node) or through other such automated processes.

In this section, we describe the use of the InstallShield and login script options.

We will focus on UNIX and NT Endpoints only. For OS/2 and Netware Endpoint installations, the process is similar. Please refer to the *TME 10 Framework Planning and Installation Guide* for details.

- InstallShield is for PCs running Windows NT, Windows 3.x, Windows 95. The images come with the TME 3.2 Framework CD and are bundled by interpreter type in `\pc\lcf`. For Windows NT, Windows 3.1, Windows 95, and Netware, you can go to the appropriate subdirectory in `\pc\lcf` and run `setup.exe`. For OS/2, run `install.exe` from the `\PC\LCF\OS2\CDROM` directories. Diskette images are also available for installation from diskette. For installation of multiple Endpoints, you can remotely mount/share the CD to the targeted Endpoints and run the install program.
- For NT, Windows95, and Windows 3.x systems, a logon script can also be used to install the Endpoints. The logon script is activated when the user logs on to a server. An example is shown later.
- For UNIX, `winstlcf` is the best way to install LCF. A file containing a list of targets can be passed to `winstlcf` if multiple Endpoints are to be



installed. It requires a UNIX shell or exec service, a Bourne-compatible shell, and a fair number of standard UNIX utilities such as grep on the target system.

### **4.3.1 Default Endpoint Login Process**

When you install LCF on a PC, the LCF daemon is automatically started which (by default) sends a broadcast to your network in search of an Endpoint Gateway. An overview of this process is described in section 3.4, "Endpoint Connection Process" on page 34. In Chapter 5, "Configuring the LCF Environment" on page 61, we look at the files created on the Endpoint and discuss this connection process in more detail.

As described in 5.3, "Controlling an Endpoint Login Through Policies" on page 64, you should set policies first before you add Endpoints. If you add Endpoints without any policy in your Endpoint Manager, the Endpoints will be assigned to Endpoint Gateways in a somewhat random fashion, and this may not lead to the most desirable configuration.

### **4.3.2 Creating a Windows NT Endpoint Using InstallShield**

This section walks through the process of installing an LCF Endpoint on NT using the InstallShield GUI.

After starting the setup program from the PC/LCF/WINNT directory on the CD the following screens are displayed. We have left a few intermediary screens out since they do not warrant any special discussion.

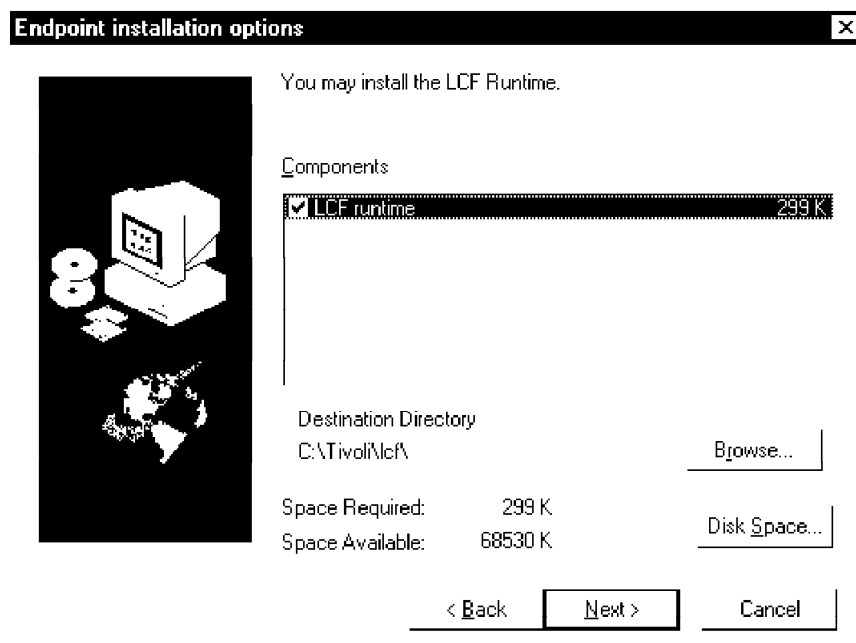


Figure 13. Initial Endpoint Installation Panel

Figure 13 shows the primary panel displayed when starting the installation of the LCF Endpoint on a Windows NT system. Note that the physical disk requirements for this platform is 299 KB.

The next panel (shown in Figure 14 on page 51) allows you to specify a specific Gateway address/port that this Endpoint should use by default. You can also specify the port that will be used on the Endpoint system for communication with the Endpoint Gateway. The **Other** field can be used to specify other parameters that are accepted on the command line when not using the GUI.

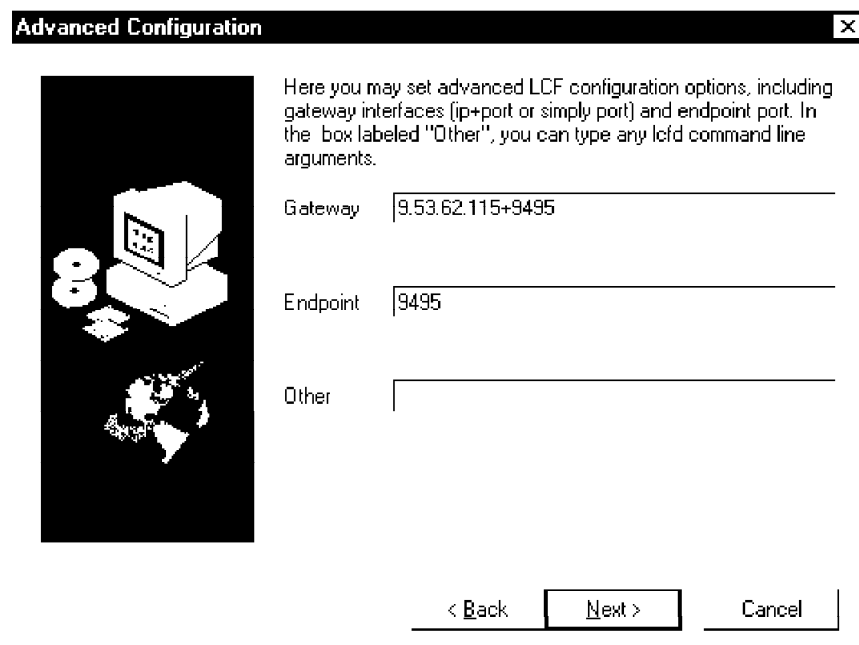


Figure 14. Endpoint Installation Parameters Panel

Once this information has been entered, the Endpoint will be installed. The default target directory is C:\TIVLI\LCF, however, it can be changed through a panel we have not shown here. Once the installation is complete, the LCF daemon is started and the panel shown in Figure 15 on page 52 is displayed to show you the status of the connection process. If everything goes as expected, you will see a message stating that the **Gateway login succeeded**.



Figure 15. Endpoint Installation Status Panel

Once you have successfully installed one or more Endpoints, we can view the Endpoint list for this Endpoint Gateway by using the Endpoint Manager and selecting the Endpoint Gateway. An example of the Endpoint List is shown in Figure 16 on page 53.

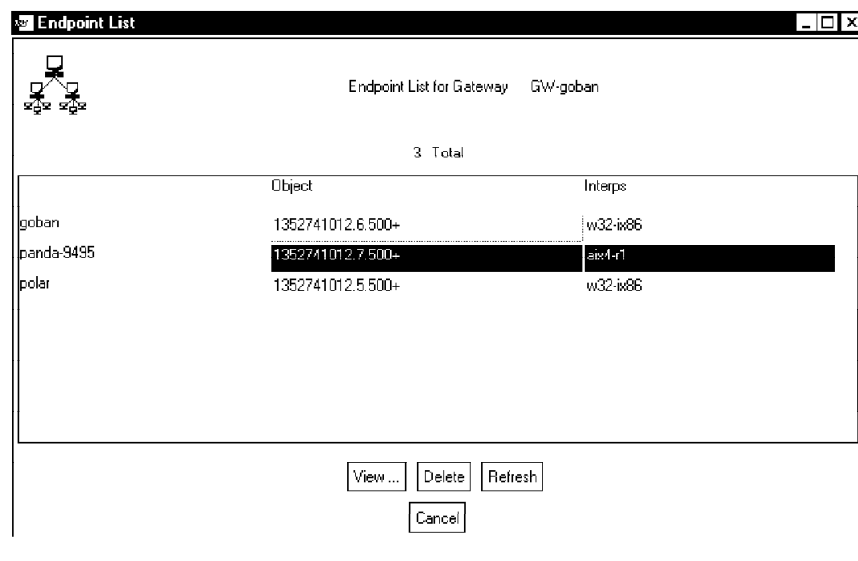


Figure 16. Gateway List after Multiple Endpoints are Installed

### 4.3.3 Installing Windows NT Endpoints from Logon Scripts

Endpoints can be automatically installed on Windows 3.x, Windows 95, and Windows NT machines using logon scripts. When a user logs on to his or her PC, the logon script checks for the Tivoli Lightweight Client service and, if necessary, launches the Endpoint installation process.

The following is an example of a logon script that will install an Endpoint. Comments are included to assist you in understanding its logic and to assist you in modifying the script for your environment.

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

```

REM Script settings - Modify these to your environment
set MAPDRIVE=v
set SERVER=\\itscfs00\tivoli
set SERVER_USER=ausres06
set INSTALDIR=\TME32\PC\LCF
set DESTDIR=c:\tivoli\lcf
set LOGDEST=c:\lcfinst.log
set ISHIELDPTS=-s

REM Inform user of log file.
echo Log file is stored in %LOGDEST%.

REM Optional introduction to user
echo Installing TME 10 Endpoint > %LOGDEST%

REM Figure out which interp
ver | find "Windows 95" > nul
if errorlevel 1 goto checkNT
set INTERP=win95
goto install

:checkNT
ver | find "Windows NT" > nul
if errorlevel 1 goto assume3x
set INTERP=winnt
goto install

:assume3x
INTERP=win3x

:install
REM Check for previous installation
net start | find "Tivoli Lightweight Client" > nul
if errorlevel 1 goto next1
if errorlevel 0 goto prevInst

:next1
REM if exist %DESTDIR%\bin\%INTERP%\nul goto prevInst

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

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

REM Execute installshield setup
start %MAPDRIVE%:%INSTALDIR%\%INTERP%\setup.exe %ISHIELDPTS%
echo Successfully installed for %USERNAME% >> %LOGDEST%
goto end

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

:end

```

#### 4.3.4 Installing UNIX Endpoints with winstlcf

A TME administrator can install Endpoints on UNIX clients from the command line using the `winstlcf` command. This command is available as soon as you install TME 10 Framework 3.2 on your TMR.

You can install as many Endpoints as necessary by either specifying machine names after the script or by using a text file as input to the `winstlcf` command.

For example, if you wish to install the UNIX machine `betty` as an Endpoint, simply execute:

```
# winstlcf betty
```

You can also install multiple UNIX machines by creating a list file. For example, assume the text file `endpoints.txt` contains a list (one system per line) of all target Endpoints. The command:

```
# winstlcf -f endpoints.txt -P
```

will install the LCF Endpoint modules on each system in the list.

The `-f` argument specifies an input file and the `-P` argument enables you to enter a global password for use on all of the machines being installed.

For a detailed explanation of the `winstlcf` command and for additional examples, refer to *TME 10 Framework Reference Manual*.

---

#### 4.4 Endpoint Data Files

The Endpoint installation creates a set of files in the LCF install directory. By default, the LCF install directory on a PC is `c:\tivoli\lcf`. On UNIX, the default installation directory is `/usr/local/Tivoli/lcf`.

The configuration and log files are placed in a directory named as follows:  
`./dat/<date-timestamp>.random_number`.

The following data files are automatically created:

- `lcf.d.log` - A text file that contains the login activities and upcall/downcall methods of the Endpoint.
- `last.cfg` - A text file that contains the Endpoint and Gateway login configuration, like port used and number of login re-tries.
- `lcf.dat` - A binary file that contains Gateway login information like hostname and IP address of the last Gateway to which the Endpoint logged in. The information contained in this file can not be modified by

the user. Certain information can be overridden by the use of command line arguments when starting the LCF daemon. However, in general, once this file is created (that is, the Endpoint has successfully logged into a Endpoint Gateway), the Endpoint should be managed via the framework mechanisms and policies, rather than by an end user adding parameters to its initialization.

- `lcf.id` - Contains additional Gateway login information.

We will focus our discussion on `LCFD.LOG` and `LAST.CFG`.

#### **4.4.1.1 LCFD.LOG**

The following figure shows the `LCFD.LOG` after a successful installation and login of an LCF Endpoint. We specified the `-d 2` parameter, which specifies a more verbose level of messages in the log file.



```

Oct 05 04:21:43 1 lcf lcf 2.1 (w32-ix86)
Oct 05 04:21:43 1 lcf run_dir: 'C:\Tivoli\lcf\dat\08221500.467\'
Oct 05 04:21:43 1 lcf logging to 'C:\Tivoli\lcf\dat\08221500.467\\lcf.log' at level 2
Oct 05 04:21:43 1 lcf cache: 'C:\Tivoli\lcf\dat\08221500.467\\cache'
Oct 05 04:21:43 1 lcf cache limit: '20480000'
Oct 05 04:21:43 1 lcf cache size at initialization: '0'
Oct 05 04:21:43 Q lcf lcf_run
Oct 05 04:21:43 2 lcf Writing GCS file: C:\Tivoli\lcf\dat\08221500.467\\last.cfg
Oct 05 04:21:43 1 lcf node_login: listener addr '0.0.0.0+9494'
Oct 05 04:21:43 2 lcf No known gateways.
Oct 05 04:21:43 2 lcf Trying other login listeners...
Oct 05 04:21:43 1 lcf Doing initial login broadcast...
Oct 05 04:21:43 Q lcf send_login_dgram: interval=300 attempts=6
Oct 05 04:21:43 Q lcf net_usend of 288 bytes to 255.255.255.255+9494. Bcast=1
Oct 05 04:21:43 Q lcf send_login_dgram: waiting for reply. attempt 1 of 6
Oct 05 04:21:43 Q lcf net_accept, handle=0x421f50
Oct 05 04:21:44 Q lcf New connection from 9.3.1.47+4369
Oct 05 04:21:45 Q lcf Entering net_recv, receive a message
Oct 05 04:21:45 Q lcf Leaving net_recv with buffer of 384 bytes, session 0
Oct 05 04:21:45 Q lcf recv: len='384' (code='14', session='0')
Oct 05 04:21:45 2 lcf Writing GCS file: C:\Tivoli\lcf\dat\08221500.467\\last.cfg
Oct 05 04:21:45 Q lcf gobantp is dispatcher 3 in region 1375598768
Oct 05 04:21:45 1 lcf Logging into new gateway...
Oct 05 04:21:45 Q lcf login_to_gw
Oct 05 04:21:45 Q lcf login_gw -> 9.3.1.47+9494
Oct 05 04:21:45 2 lcf Connecting to '9.3.1.47+9494'
Oct 05 04:21:45 Q lcf net_send of 384 bytes, session 3
Oct 05 04:21:45 Q lcf net_accept, handle=0x421f50
Oct 05 04:21:45 Q lcf New connection from 9.3.1.47+4370
Oct 05 04:21:45 Q lcf Entering net_recv, receive a message
Oct 05 04:21:45 Q lcf Leaving net_recv with buffer of 456 bytes, session 3
Oct 05 04:21:45 Q lcf recv: len='456' (code='14', session='3')
Oct 05 04:21:45 2 lcf Writing GCS file: C:\Tivoli\lcf\dat\08221500.467\\last.cfg
Oct 05 04:21:45 Q lcf gobantp is dispatcher 3 in region 1375598768
Oct 05 04:21:45 1 lcf write login file 'lcf.dat' complete
Oct 05 04:21:46 1 lcf final pid: 260
Oct 05 04:21:46 1 lcf Login to gateway complete.
Oct 05 04:21:46 1 lcf Ready. Waiting for requests (0.0.0.0+9494).
Oct 05 04:21:46 2 lcf Run timeout set: 120.
Oct 05 04:21:46 Q lcf Entering Listener
Oct 05 04:21:46 Q lcf net_wait_for_connection, handle=0x421f50

```

Figure 17. LCFD.LOG File After Installation

When the LCF daemon installs and starts, it creates the working directory and creates the log file.

```
Oct 05 04:21:43 1 lcf lcf 2.1 (w32-ix86)
Oct 05 04:21:43 1 lcf run_dir: 'C:\Tivoli\lcf\dat\08221500.467\'
Oct 05 04:21:43 1 lcf logging to 'C:\Tivoli\lcf\dat\08221500.467\lcf.log' at level 2
```

It then creates a cache directory where the Endpoint will process methods from the Gateway.

```
Oct 05 04:21:43 1 lcf cache: 'C:\Tivoli\lcf\dat\08221500.467\cache'
Oct 05 04:21:43 1 lcf cache limit: '20480000'
Oct 05 04:21:43 1 lcf cache size at initialization: '0'
```

The LCF daemon starts to run and checks if there is an existing configuration file (last.cfg). Being a new install, it immediately creates an initial last.cfg file.

```
Oct 05 04:21:43 Q lcf lcf_run
Oct 05 04:21:43 2 lcf Writing GCS file: C:\Tivoli\lcf\dat\08221500.467\last.cfg
```

Now, here is where the network broadcast occurs. It sends a login packet within its subnet in search for an Endpoint Gateway listening at port 9494 (255.255.255.255+9494). By default, it makes 6 attempts at an interval of 5 minutes.

```
Oct 05 04:21:43 2 lcf No known gateways.
Oct 05 04:21:43 2 lcf Trying other login listeners...
Oct 05 04:21:43 1 lcf Doing initial login broadcast...
Oct 05 04:21:43 Q lcf send_login_dgram: interval=300 attempts=6
Oct 05 04:21:43 Q lcf net_usend of 288 bytes to 255.255.255.255+9494. Bcast=1
Oct 05 04:21:43 Q lcf send_login_dgram: waiting for reply. attempt 1 of 6
Oct 05 04:21:43 Q lcf net_accept, handle=0x421f50
```

We receive a connection from the Endpoint Gateway (9.3.1.47). The information about our newly assigned Endpoint Gateway is written to last.cfg and we receive information about our assigned dispatcher number and region identifier.

```
Oct 05 04:21:44 Q lcf New connection from 9.3.1.47+4369
Oct 05 04:21:45 Q lcf Entering net_recv, receive a message
Oct 05 04:21:45 Q lcf Leaving net_recv with buffer of 384 bytes, session 0
Oct 05 04:21:45 Q lcf recv: len='384' (code='14', session='0')
Oct 05 04:21:45 2 lcf Writing GCS file: C:\Tivoli\lcf\dat\08221500.467\last.cfg
Oct 05 04:21:45 Q lcf gobantp is dispatcher 3 in region 1375598768
```

Information is written to lcf.dat, our login is completed and the Endpoint starts listening for any downcalls that may arrive.

```
Oct 05 04:21:45 1 lcf write login file 'lcf.dat' complete
Oct 05 04:21:46 1 lcf final pid: 260
Oct 05 04:21:46 1 lcf Login to gateway complete.
Oct 05 04:21:46 1 lcf Ready. Waiting for requests (0.0.0.0+9494).
Oct 05 04:21:46 2 lcf Run timeout set: 120.
Oct 05 04:21:46 Q lcf Entering Listener
Oct 05 04:21:46 Q lcf net_wait_for_connection, handle=0x421f50
```

As operations are performed on the Endpoint (that is, downcalls arrive), this log will contain information about those calls.

After the first login, subsequent logins will have even simpler entries in the log, since the Endpoint will not have to broadcast to find an Endpoint Gateway.

#### 4.4.1.2 LAST.CFG

Let's look at the file `last.cfg`. This file contains configuration information from the last successful login to a Endpoint Gateway. This information will normally be used on subsequent logins to avoid the broadcast mechanism for locating an Endpoint Gateway. Although, as we will see in the next chapter, a different file (`lcf.cfg`) using the same format, can define static configuration data that does not change with each successful login. If `lcf.cfg` exists, it will be used in place of `last.cfg`. Additional information, including login credentials are stored in the binary file, `LCF.DAT`.

```
lcf_port=9494
gateway_port=9494
log_threshold=2
start_timeout=120
run_timeout=120
lcf_version=2.1
logfile=C:\Tivoli\lcf\dat\08221500.467\\lcf.log
config_path=C:\Tivoli\lcf\dat\08221500.467\\last.cfg
run_dir=C:\Tivoli\lcf\dat\08221500.467\
load_dir=C:\Tivoli\lcf\bin\w32-ix86\mrt\
lib_dir=.
cache_loc=C:\Tivoli\lcf\dat\08221500.467\cache
cache_index=Index.v14
cache_limit=20480000
log_queue_size=1024
log_size=1024000
udp_interval=300
udp_attempts=6
```

Figure 18. LAST.CFG File

Most of these values entries are self evident. The most important values here are:

- `lcf_port` - The port used by the Endpoint to communicate with the Endpoint Gateway
- `gateway_port` - The listening port of the Endpoint Gateway
- `udp_interval` - Login retry intervals in seconds
- `udp_attempts` - Number of login retries

In Chapter 5, “Configuring the LCF Environment” on page 61, we’ll talk about using this file as a model for `lcf.d.cfg` for providing a customized configuration file for your Endpoints.

---

## 4.5 Summary

In this chapter we have described the installation procedure for LCF Endpoint Managers, Endpoint Gateways and Endpoints. In addition, we briefly reviewed the log file and configuration file that are created during an Endpoint’s first successful login to an Endpoint Gateway.

In the next chapter, we will continue talking about the configuration of the LCF environment to help manage the association between Endpoints and Endpoint Gateways.

---

## Chapter 5. Configuring the LCF Environment

In the classic TME environment (TMR Servers, Managed Nodes and PC Managed Nodes), the installation process implicitly determined the TMR to which a managed system belonged and there was no requirement to explicitly configure managed nodes to communicate with their TMR server.

In an LCF environment, Endpoints must be configured (manually or through self-configuration) to communicate with an appropriate Endpoint Gateway. To allow for both flexibility as well as availability, the association between an Endpoint and a Gateway can be dynamically determined and may change if a particular Gateway is not available.

This chapter describes how you, as an administrator, can control the configuration of your LCF environment to control the associations between Endpoints and Gateways. There are two primary methods that can be used; Endpoint configuration files and policies.

---

### 5.1 Controlling an Endpoint Login

By default, a newly installed Endpoint, issues a broadcast message looking for an Endpoint Gateway it can use to connect to a TMR. Gateways that receive this broadcast, pass this login request to the Endpoint Manager who chooses a Gateway. That Gateway is then notified, and the Gateway then contacts the Endpoint to initiate the actual login. By default, the TMR Server chooses the first Endpoint Gateway that has reported the broadcast login request from an Endpoint.

In an environment with multiple Gateways or multiple TMRs, it is highly desirable to provide controls that limit which Endpoint Gateways may be chosen for a new Endpoint.

There are currently two ways of controlling how an Endpoint becomes associated with an Endpoint Gateway.

1. Endpoint Configuration Files - When the Endpoint is started, it first looks for the existence of various configuration files. If these files exist, it will use information within these files that identify a specific Endpoint Gateway. The Endpoint will then try to login to this Endpoint Gateway without issuing a broadcast message. If the Endpoint Gateway specified in the configuration files does not respond, then the Endpoint will resort to the broadcast mechanism searching for a different Endpoint Gateway.
2. Endpoint Policies - Policies (implemented through policy scripts) can be put in place to control how the Endpoint Manager and Endpoint

Gateways handle login requests from Endpoints. Implementing proper Endpoint policies can be very effective in large networks with many Endpoints, as it can help avoid the requirement to distribute configuration files to newly installed Endpoints.

---

## 5.2 Controlling an Endpoint Login Through Configuration Files

In section 4.4, “Endpoint Data Files” on page 55 we described the various files that are created on an Endpoint after a successful login to a TMR.

These files are reviewed below:

- `lcf.dat` - A binary file containing information related to the Endpoint Gateway assigned to the Endpoint. The information contained in this file can not be modified by the user. Certain information can be overridden by the use of command line arguments when starting the LCF daemon. However, in general, once this file is created (that is, the Endpoint has successfully logged into an Endpoint Gateway), the Endpoint should be managed via the framework mechanisms and policies, rather than by an end user adding parameters to its initialization.
- `last.cfg` - Contains the configuration information used the last time the Endpoint was started. This file is automatically created upon installation and modified each time the Endpoint logs in.
- `lcf.d.log` - Contains Endpoint messages like log-in activities.
- `lcf.id` - Contains additional Gateway login information.

An additional file that can be used to affect the configuration of your environment, but not created for you, is `lcf.d.cfg`.

This file has the identical format and content as `last.cfg`. However, it is not modified after each login, so it does not necessarily contain information related to the last successful login. Therefore it can be used to define static configuration data that is not modified after each successful login. For instance, if we were to modify `last.cfg` to specify a preferred Endpoint Gateway, and that Endpoint Gateway was temporarily unavailable, then the Endpoint would broadcast for a new Endpoint Gateway and if successful, the file would be modified to point to the new Endpoint Gateway. The next time the Endpoint was restarted, it would then attempt to attach to the last Endpoint Gateway it successfully used.

The LCF daemon will use information stored in `lcf.dat` if this file exists, otherwise it will use `lcf.d.cfg` if it exists, and finally will use `last.cfg`. However, as already mentioned, LCF Endpoints are not designed with the intention of end users modifying their operation. They are designed to

locate an Endpoint Gateway during their initialization and from that point on they should be controlled through the framework and policies.

If you have a particular need to control an Endpoint's configuration in a more manual way, then you can erase `lcf.dat` and create a file called `lcf.d.cfg` as shown in Figure 19.

However, you need to be aware that Endpoint specific information is stored in `lcf.dat`, and by erasing it, a new Endpoint object is created within the TMR the next time the Endpoint connects. Therefore, the old Endpoint object will need to be removed to avoid confusion on the part of the administrators.

It should also be noted, that even if you specify a specific Endpoint Gateway by supplying the `lcs.login_interfaces` parameter, this does not guarantee that the specified system will be your Endpoint Gateway. Rather, the Endpoint will attempt to contact it to start the login process and the policies of the Endpoint Manager will determine the ultimate Endpoint Gateway for the Endpoint.

```
lcf_port=1027
gateway_port=9494
bcast_addr=255
log_threshold=2
start_timeout=120
run_timeout=120
lcf_version=2.1
logfile=C:\Tivoli\lcf\dat\06191422.307\\lcf.d.log
config_path=C:\Tivoli\lcf\dat\06191422.307\\last.cfg
run_dir=C:\Tivoli\lcf\dat\06191422.307
load_dir=C:\Tivoli\lcf\bin\w32-ix86\mrt\
lib_dir=C:\Tivoli\lcf\bin\w32-ix86\mrt\
cache_loc=C:\Tivoli\lcf\dat\06191422.307\\cache
cache_index=Index.v13
cache_limit=20480000
log_queue_size=1024
log_size=1024000
udp_interval=300
udp_attempts=6
#lcs.login_interfaces=<gw_addr><gw_port>
lcs.login_interfaces=9.3.1.235+9494
#lcs.machine_name=<my_Endpoint_name>
lcs.machine_name=bambam
```

Figure 19. Sample `lcf.d.cfg` File

The last four lines of this configuration file are added to force the Endpoint to attempt to log in to an Endpoint Gateway with IP address 9.3.1.235 and communicate on port number 9494. The Endpoint machine name is also specified. The rest of the information is the same as would be contained in a `last.cfg` file.

For information describing each of the entries and its usage, please refer to the documentation for the `lcmd` command in *TME 10 Framework Reference Manual*.

---

## 5.3 Controlling an Endpoint Login Through Policies

There are four types of policies that can be put in place for controlling Endpoint logins to a TMR. Three of these policies are implemented on the Endpoint Manager and one on the Endpoint Gateway.

The Endpoint Manager policies are called:

- **allow\_install\_policy**
- **select\_gateway\_policy**
- **after\_install\_policy**

These policies run in the order listed above. The Endpoint Gateway implements a policy called **login\_policy**.

Information on each of these policies is presented in the following sections.

### 5.3.1 allow\_install\_policy

The **allow\_install\_policy** script is executed whenever an Endpoint's login request packet is received at the Endpoint Manager. This packet is broadcast by the Endpoint, received by the Endpoint Gateway, and then communicated to the Endpoint Manager via `oserv-oserv` communications. When the Endpoint Manager receives this information, the **allow\_install\_policy** script is executed to determine whether or not this Endpoint should be allowed to login to this TMR. By default, all login requests are granted.

This policy is very useful in an environment with multiple TMRs. For instance, you could create a text file containing the names of Endpoints allowed in a particular TMR and use this list as an input to the policy. An example of such a policy is shown below.



### 5.3.1.1 Example

In our scenario, we have two TMRs on the same subnet - Flintstones and Simpsons. There are four Endpoints that are about to login and without any active policy, the Endpoint Manager in each TMR will receive the login request and assign the Endpoint to one of its Endpoint Gateways. So which TMR does each Endpoint actually belong to?

The first TMR to call back to the Endpoint will initially claim the Endpoint for its own. The Endpoint will then complete its login and the proper encryption keys will be set to allow the first TMR access to the Endpoint. However, in the current implementation (which may change before you read this), when the second TMR responds, the Endpoint (and its encryption key) will be reset. Therefore, only the second TMR will be able to manage this Endpoint. As hinted above, work is being done to resolve this in a cleaner fashion. The proper way of handling this today, and maybe even after the implementation changes, is to control this through policies on the Endpoint Manager.

To control the Endpoint login, you can use the script below to enable some filtering before an Endpoint is accepted as a member of the TMR. This script compares the name of the Endpoint system (passed as a parameter) to entries in a list of host names included in a file called `eplist.txt`. If the Endpoint name matches an entry in the list, the script exits with a value of 0. Otherwise it exits with a value of 1.

So for the Flintstones TMR, the allowed Endpoints might be `pebbles` and `bambam`. While on the Simpsons TMR, the allowed Endpoints might be `homer` and `lisa`. The `eplist.txt` used for the Flintstones TMR server might look like the following:

```
[root@panda]/var/spool/Tivoli/policy/lcf> cat eplist.txt
bambam
pebbles
```

The `allow_install_policy` script that enforces our policy is:

```
#!/bin/sh

#
# 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 id of the Endpoint machine
# $3 - The architecture type of the Endpoint machine
# $4 - The object id of the gateway that the Endpoint logged into
```

```

# $5 - The ip address of the Endpoint logging in.
#

# 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/../policy/lcf/eplist.txt

exit 1

```

### 5.3.2 select\_gateway\_policy

The **select\_gateway\_policy** is run after the **allow\_install\_policy**. It runs on the Endpoint Manager each time it receives a login request packet from an Endpoint. It determines the Endpoint Gateway to which an Endpoint can log in. The login packet is sent either on initial login or on subsequent logins when the Endpoint has lost contact with its assigned Endpoint Gateway.

If multiple Endpoint Gateways meet the criteria specified in the script, **select\_gateway\_policy** can return a list of possible Endpoint Gateways. The Endpoint Manager attempts to contact the first Endpoint Gateway on the list. If the Endpoint Manager fails to contact the Endpoint Gateway, it continues in order through the list until it successfully contacts an Endpoint Gateway. The first Endpoint Gateway contacted is the Endpoint Gateway to which the Endpoint is assigned. If the script fails, the Endpoint Manager's default selection criteria is used. The failure of the script does not stop the login process.

#### 5.3.2.1 Example

The following is a C program that could be used to implement the **select\_gateway\_policy**. This example is extracted from the *TME 10 Framework Reference Manual*. It generates a list of all Endpoint Gateways on the same subnet as the Endpoint. The Endpoint Manager will attempt to contact each Endpoint Gateway returned until successful and assign the Endpoint to it.

If the policy fails, or if no Endpoint Gateways meet the criteria, then the Endpoint Manager uses its default selection, which is to use the first Endpoint Gateway to report the Endpoint's login request.

```

#include <tivoli/defines.h>
#include <tivoli/ExException.h>
#include <tivoli/tas_init.h>
#include <tivoli/dir.h>
#include <tivoli/TNR.h>
#include <tivoli/t_man_node.h>
#include <tivoli/sequence.h>
#include <gateway/t_Gateway.h>
#include <stdio.h>
int main (int argc, char **argv)
{
    type_repository *t[1];
    TMF_TNR_ObjectInfoList gws, mns;
    TMF_ManagedNode_net_drop_list_t ifs;
    Environment ev = {0};
    char *ep_label, *ep_oid, *ep_interp, *gateway, *ep_ip;
    int i, j, if_ctr;
    Object mn_oid;
    char *mn, *mn_ip;
    t[0] = type_repository_null;

    /* necessary to be a useful TME client */
    tmf_init(t);
    tmf_client_init();
    tas_init();

    /* get all the cli args */
    ep_label = argv[1];
    ep_oid = argv[2];
    ep_interp = argv[3];
    gateway = argv[4];
    ep_ip = argv[5];
    Try {
        /* we just want the subnet of the endpoint */
        i = strlen( ep_ip );
        while( ep_ip[i] != '.' )
            i--;
        ep_ip[i] = '\0';

        /* get all the gateways and search for the one
           that is on the same subnet that the endpoint
           is on.
        */
        gws = dir_get_all_instances("Gateway");
        for( i=0; i<gws._length; i++)
        {
            /* Find the ip address of the machine the
               gateway is running on */
            Try {
                mn = t_TMF_Gateway_Gateway__get_proxy(
                    gws._buffer[i].object,
                    &ev, Trans_none );
                mn_oid = dir_lookup_instance(
                    "ManagedNode", mn );

                ifs =
                    t_TMF_ManagedNode_Managed_Node_list_ip_interfaces(
                        mn_oid, &ev, Trans_none );
                for( if_ctr=0; if_ctr<ifs._length; if_ctr++)
                {
                    /* need the interface that TME is using. */

```

```

        if( ifs._buffer[ if_ctr ].used_by_oserv == 1 )
        {
            mn_ip = mg_strdup(
                ifs._buffer[if_ctr].address );
            break;
        }
    }
    /* compare the subnet of the gateway with that
    of the endpoint
    */
    j = strlen( mn_ip );
    while( mn_ip[j] != '.' )
        j--;
    mn_ip[j] = '\0';
    if( !strcmp( ep_ip, mn_ip ) )
        printf("%s", gws._buffer[i].object );
    mg_free( mn_ip );
    mg_free( mn );
    mg_free( mn_oid );
    seq_free_buffer( &ifs );
} CatchAll() {
    /* If we got here, the gateway is down
    and is not viable anyways. */
}
EndTry;
}
seq_free_buffer(&gws);
} Catch(Exception, ex) {
    fprintf(stderr, "%s\n",
        tmf_ex_msg_bind(ex, NULL, 0));
    exit( 1 );
}
EndTry;
exit(0);
}

```

Note that the list of Endpoints is simply printed to stdout via the printf() function. The Endpoint Manager will receive this data and use it to find the first active Endpoint Gateway and assign it the Endpoint to it.

### 5.3.3 after\_install\_policy

After the Endpoint has successfully logged in to the TMR for the first time, you may want to run scripts or tasks to perform functions such as adding the Endpoint as a subscriber to a profile manager. The **after\_install\_policy** is where you put these tasks. This policy runs only after the initial login. It does not run on subsequent logins.

Since this Endpoint already exists when this script runs, the failure of this script does not stop the login process.

#### 5.3.3.1 Example

The script below runs after a successful Endpoint login. It subscribes new Endpoints of similar architecture types to specific profile managers. If the policy region or profile manager does not exist, the policy creates them.

Comments are included within this script to explain its various steps.

```
#!/bin/sh
#
# 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 id of the Endpoint machine
# $3 - The architecture type of the Endpoint machine
# $4 - The object id of the gateway that the Endpoint logged into
# $5 - The ip address of the Endpoint logging in.
#

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 the administrator's
# 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"
    wrtpr -m ProfileManager -a $ADMIN $LCF_POLICY_REGION
    idlcall $ADMIN_OID refresh_collection
fi

# Check to see if our interp specific profile manager already
# exists. If not, create it and make it dataless so that we
# 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
    wcrtpfmgr $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

```

### 5.3.4 login\_policy

The **login\_policy** script is executed each time an Endpoint logs in to its assigned Endpoint Gateway. The policy is run by the Endpoint Gateway that the Endpoint is logging in to.

If the **login\_policy** runs successfully, the Endpoint is allowed to log in. If the script fails, the login process is terminated.

Therefore, this policy can be used to both restrict access, by not allowing the login to proceed, or to perform other functions at login time. For example, you may wish to post a notice to a notice group when certain Endpoints login to the TMR, as shown in the example below.

#### 5.3.4.1 Example

The following example logs a notice to an LCF related notice group every time an Endpoint logs in.

```

#!/bin/sh
#
# The following are the command line arguments passed to this script
# from the Gateway.
#
# $1 - The label of the Endpoint machine
# $2 - The architecture type of the Endpoint machine
# $3 - The object id of the gateway that the Endpoint logged into
# $4 - The IP address of the Endpoint logging in
#
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 $3 _get_label
EPOID=wlookup -o -r Endpoint $1

wsndnotif $1 ($EPOID of inter type, $2, logged into gateway $GW ($3).
LCF_NOTICE

exit 0

```

### 5.3.5 Applying Endpoint Policies

The TME 10 Framework is installed with default Endpoint policy scripts. These do not contain any logic, and simply return control back to the Endpoint Manager or Endpoint Gateway.

To implement meaningful policies for your environment, you must replace the default policies with your own scripts. Doing this, requires three basic steps:

- Retrieve the current policy script using the `wgeteppol` command.
- Modify this script to add the appropriate logic.
- Replace the current policy script with the new script using the `wputeppol` command.

These commands take a single parameter that specifies the policy whose script you are retrieving or replacing. The script itself is routed to stdout for the `wgeteppol` command or read from stdin for the `wputeppol` command. Therefore you will normally use redirection operators as shown in our example below.

For additional information on the `wgeteppol` and `wputeppol` commands, refer to the *TME 10 Framework Reference*.

### 5.3.5.1 Example

Let's assume that we want to put in place a customized **allow\_install\_policy**. We can login to the TMR and issue the following command:

```
# wgetepool allow_install_policy > allow_install_policy.sh
```

The above command will create a file called `allow_install_policy.sh` with the contents of the current policy script. In the case of a newly installed TMR with the default policies still in place, the `allow_install_policy.sh` file will contain the following:

```
#!/bin/sh

#
# 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 id of the endpoint machine
#     $3     - The architecture type of the endpoint machine
#     $4     - The object id of the gateway that the endpoint logged into
#     $5     - The ip address of the endpoint logging in.
#

exit 0
```

Figure 20. Default `allow_install_policy` Script

This file can then be edited to include logic such as that shown in section 5.3.1.1, "Example" on page 65.

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

```
# wputepool allow_install_policy < allow_install_policy.sh
```

All policy scripts are stored in the TME 10 database. The Endpoint Manager will ensure that all Endpoint Gateways receive the latest **login\_policy** script. Note that the same **login\_policy** script applies to all Endpoint Gateways in the TMR. If logic specific to the operating system running on a Endpoint Gateway is required, then the **login\_policy** script must have the logic within it to determine on which platform it is executing and act accordingly.

We should note that the `wputepool` command is used in the same way whether the policy being enabled is a shell script or a binary (compiled executable). If using a compiled executable, you must ensure that it is



compatible with the specific Endpoint Manager it will execute on. In the case of the **login\_policy** where the same policy executes on all Endpoint Gateways the you can only implement this policy as a binary of all Endpoint Gateways are running the same operating system.

---

## 5.4 Summary

Configuration of an LCF environment can be controlled by a combination of configuration files and policy scripts. This chapter has provided examples of both.

The configuration process is not terribly complex nor difficult. Proper planning for the design and implementation of your TMR is critical to a successful implementation. Once decisions related to the desired number of Endpoint Gateways, the allocation of Endpoints across Endpoint Gateways, and the communication parameters (such as ports and timeout value) have been made, the proper configuration files and policy scripts can be put in place to implement the desired TME 10 environment.



---

## Chapter 6. TME 10 Framework Web Interface

As a precursor to future capabilities to perform management functions via a World Wide Web browser, V3.2 of the TME 10 Framework provides capabilities in this area.

TME Servers, Managed Nodes and LCF Endpoints now include built-in HTTP daemons (web servers) that can be accessed from your favorite web browser. (The daemons on Managed Nodes can not be accessed directly, but only via the server acting as a proxy. More on this soon.) Both of these web servers are designed so as not to interfere with other web servers you might be running on the same system. To access the TME HTTP daemon, you simply use a URL that consists of the system name (or IP address) and TME 10's port number. For TMR Servers, the port number defaults to 94 and for LCF Endpoints, the default is 9494.

The HTTP daemon on the TMR Server is meant as a general purpose web server for TME 10 applications. We will describe this in more detail in section 6.1, "Accessing the TMR Server."

The HTTP daemon on LCF Endpoints is not a general purpose server, but, rather, provides specific information and configuration capabilities for the LCF Endpoint. These services will be described in section 6.2, "LCF Endpoint Web Server" on page 78.

---

### 6.1 Accessing the TMR Server

The web server function for the TMR Server is designed to be used by TME 10 applications. TME 10 applications will be able to use this service to present a web-based interface for administrators. Currently, the framework itself does not use this service to allow administrators to perform framework-specific management functions.

Figure 21 on page 76 shows what you might see after installing the TMR Server, and accessing it with your web browser.



Figure 21. Primary Web Page on a TMR Server

As applications that utilize this service are installed, a menu will be presented that will allow an authorized administrator to perform application-specific functions.

### 6.1.1 Starting and Stopping the HTTP Daemon

There are two new commands that can be used to start and stop the HTTP daemon on the server:

- wstarthttpd
- wstophttpd

### 6.1.2 Accessing the TMR Server

Once installed, the web server (HTTP daemon) uses an arbitrary and unused port on the system. The `oserv` daemon will route any HTTP requests arriving at the `oserv`'s port (default is 94) to the actual port in use by the HTTP daemon. Therefore, the user does not need to know the actual port used by the daemon. In fact, on subsequent initializations of the server, the actual port used may change.

To access the TMR Server, you would use a URL such as  
`http://panda.itsc.austin.ibm.com:94` //for a TMR Server

### 6.1.3 Security

The HTTP daemon supports authentication realms to allow a secure level of access to the TME 10 functions. The security is directory-based. The authentication program determines whether a particular user can access the contents of a directory. The mechanism uses a 'basic' authentication scheme using a base64-encoded username and password.

There are three new commands to allow the administrator to manage authentication realms. These commands are:

- `waddrealm` // Used to add a new realm
- `wdelrealm` // Used to delete a realm
- `wlsrealms` // Used to list current realms

See the *TME 10 Framework Reference Manual* for detailed information on these commands.

### 6.1.4 Accessing Managed Nodes

As previously mentioned, the HTTP daemon will not allow direct access to Managed Nodes. You can access the daemon running on a TMR Server. The Server will act as a proxy server for Tivoli HTTP daemons running on Managed Nodes. Therefore, if an application requires a CGI program to execute on a Managed Node, it will route requests to the appropriate Managed Node, transparently to the user.

## 6.1.5 Adding Your Own Information

**Note:** The facilities described here are primarily for use by application developers wishing to take advantage of the web server function in the TME 10 Framework. Normally, these facilities would not be used by TME 10 administrators.

The files that will be served by the HTTP daemon are located in the following directory:

```
/usr/local/Tivoli/bin/generic/HTTPd
```

If we consider the above directory to be `HTML_DIR`, then:

The default web page is `$HTML_DIR/Default.html`

Images are accessed from `$HTML_DIR/images`

Java classes are located in `$HTML_DIR/classes.`, and

On-line HTML help files are located in `$HTML_DIR/Help`

You should not modify the `Default.html` file directly. Instead, there is a command called `wupddefhtml` which will search for all registered instances of a class called `DefaultHTMLItems` and add these items to the default page.

The HTTP daemon included with the TME 10 Framework supports HTTP V1.0 and CGI V1.1 protocols. The daemon will run CGI programs that have been placed in `$BINDIR/TAS/HTTPd/cgi-bin`.

You can add your own files to the above directories and access them explicitly or through items added to the default page.

## 6.1.6 Logs

There are two useful logs you can use to investigate problems that occur accessing your web pages and CGI scripts. These logs are:

- `$DBDIR/.HTTPd/httserv.log` - This log lists information about CGI programs that have been executed.
- `$DBDIR/.HTTPd/httptran.log` - This log lists all incoming HTTP requests.

---

## 6.2 LCF Endpoint Web Server

Unlike the HTTP daemon that is available on Managed Nodes, the LCF Endpoint's web server provides very specific information and services to the administrator. An administrator can use a web browser to query information about the LCF Endpoint as well as use it to change its configuration parameters. There are seven specific pages that display the information and allow you to alter the Endpoint's configuration parameters.

In this section we show you these screens and explain those items that are not self-evident.

### **6.2.1 Accessing the LCF Daemon Status Page**

Much like we did to access the TMR Server's page, we simply access the LCF Endpoint's first page by using the Endpoint's network name and the port it uses for communication with the Endpoint Gateway. Should you forget, this port defaults to 9494.

Therefore, to access an LCF Endpoint known as gobantp we use our web browser to access:

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

Our web browser now displays the page shown in Figure 22 on page 80.

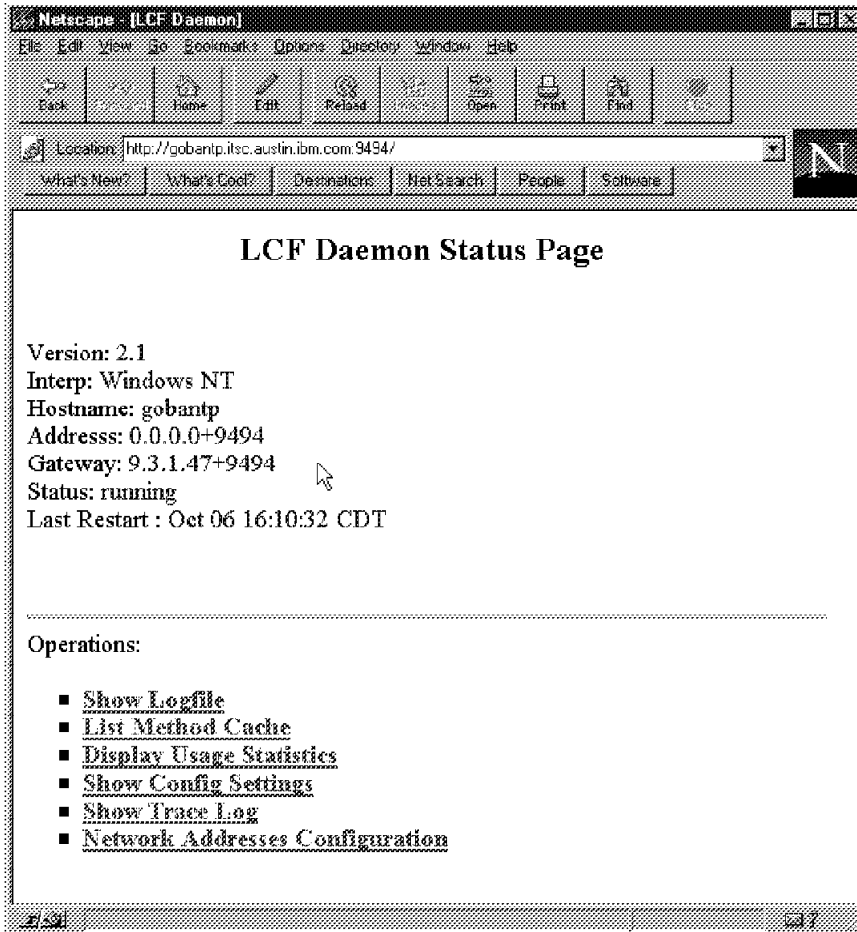


Figure 22. LCF Endpoint's Home Page

This top half of this page provides information regarding the LCF Endpoint, including:

- **Version** - The version of the LCF Endpoint code (currently 2.1)
- **Interp** - The operating system on the LCF Endpoint
- **Hostname** - The Endpoint's hostname
- **Address** - The address will always show 0.0.0.0+<port number>
- **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



The bottom half of this screen provides links to several pages providing additional information and configuration options. Each of these pages are described in the following sections.

## 6.2.2 Logfile Page

This page simply displays the lcfld.log file that was described in section 4.4, “Endpoint Data Files” on page 55.

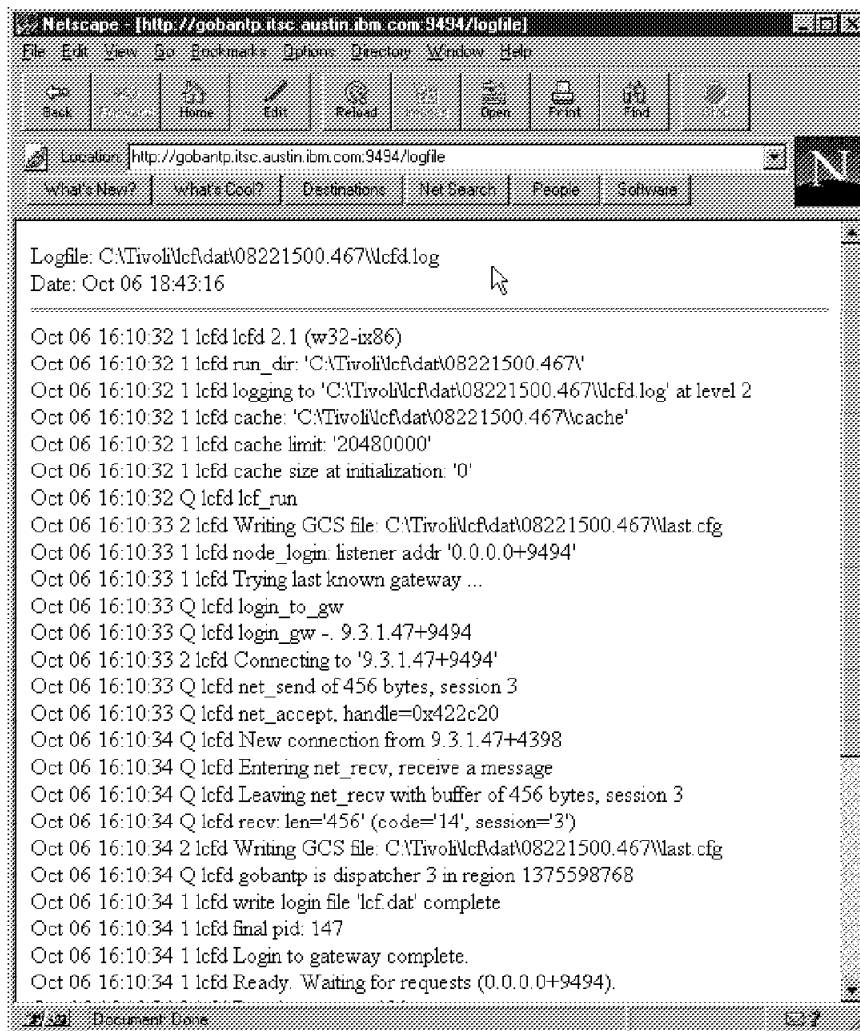
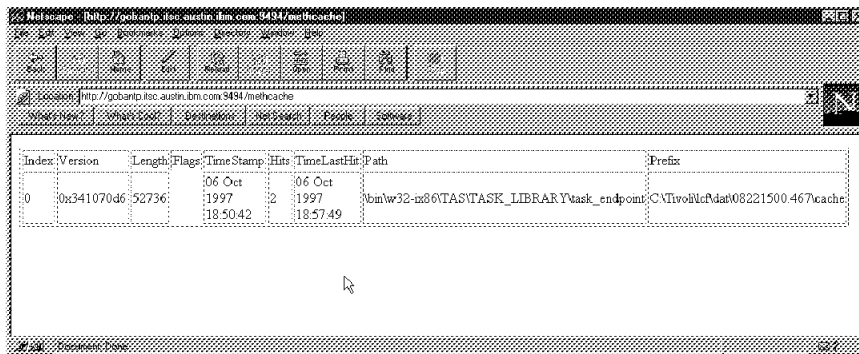


Figure 23. LCF Endpoint's Logfile Page

### 6.2.3 Method Cache Page

It is sometimes useful to be able see a listing of the methods that are currently available in an Endpoint's method cache. This page shown in Figure 24 provides that capability.



Index	Version	Length	Flags	Time Stamp	Hits	Time Last Hit	Path	Prefix
0	0x341070d6	52736		06 Oct 1997 18:50:42	2	06 Oct 1997 18:57:49	bin\w32-m86\TAS\TASK_LIBRARY\task_endpoint	C:\Ivob\lcpdat\08221500.467\cache

Figure 24. LCF Endpoint's Method Cache Page

In this case we have only a single method in our cache. This is the method that was used to execute a TME 10 task on our LCF Endpoint.

### 6.2.4 Usage Statistics Page

This page, shown in Figure 25 on page 83, 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.

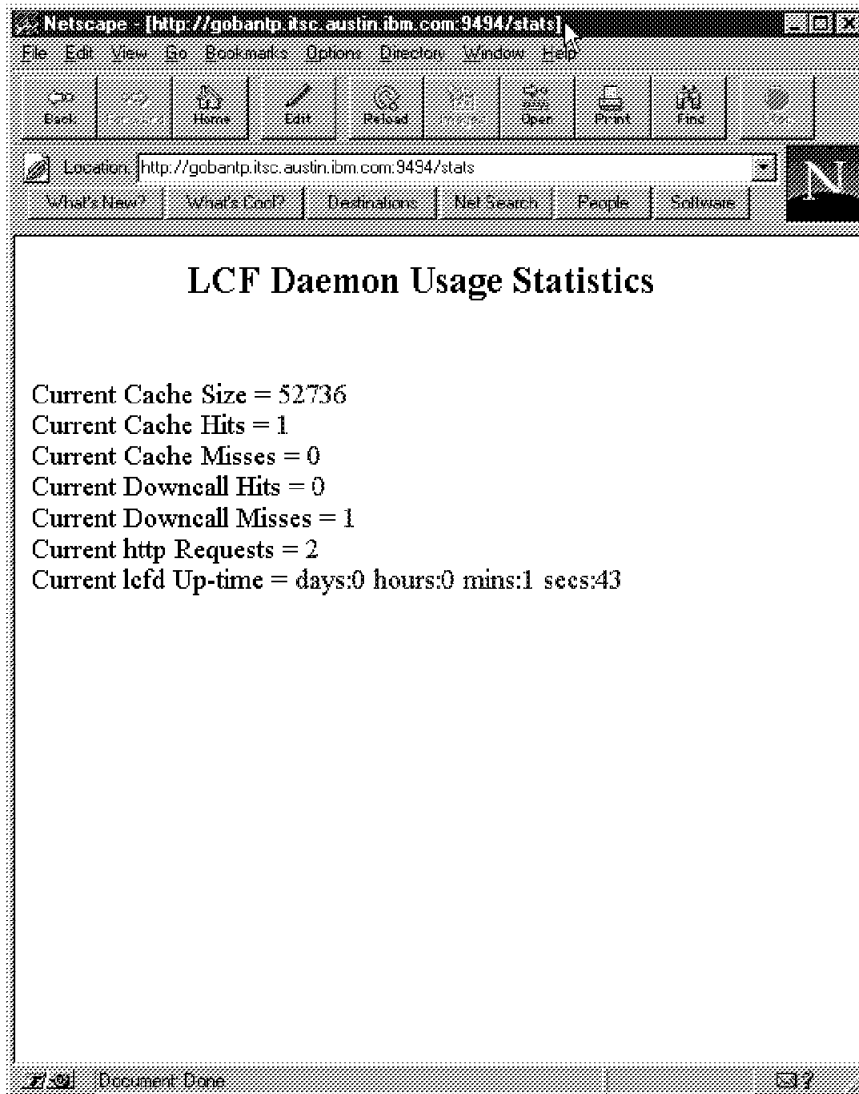


Figure 25. LCF Endpoint's Usage Statistics Page

### 6.2.5 Configuration Settings Page

This page, shown in Figure 26 on page 84, simply displays the current configuration settings for the Endpoint. The information presented is made up of information contained in the last.cfg file and lcf.dat file.



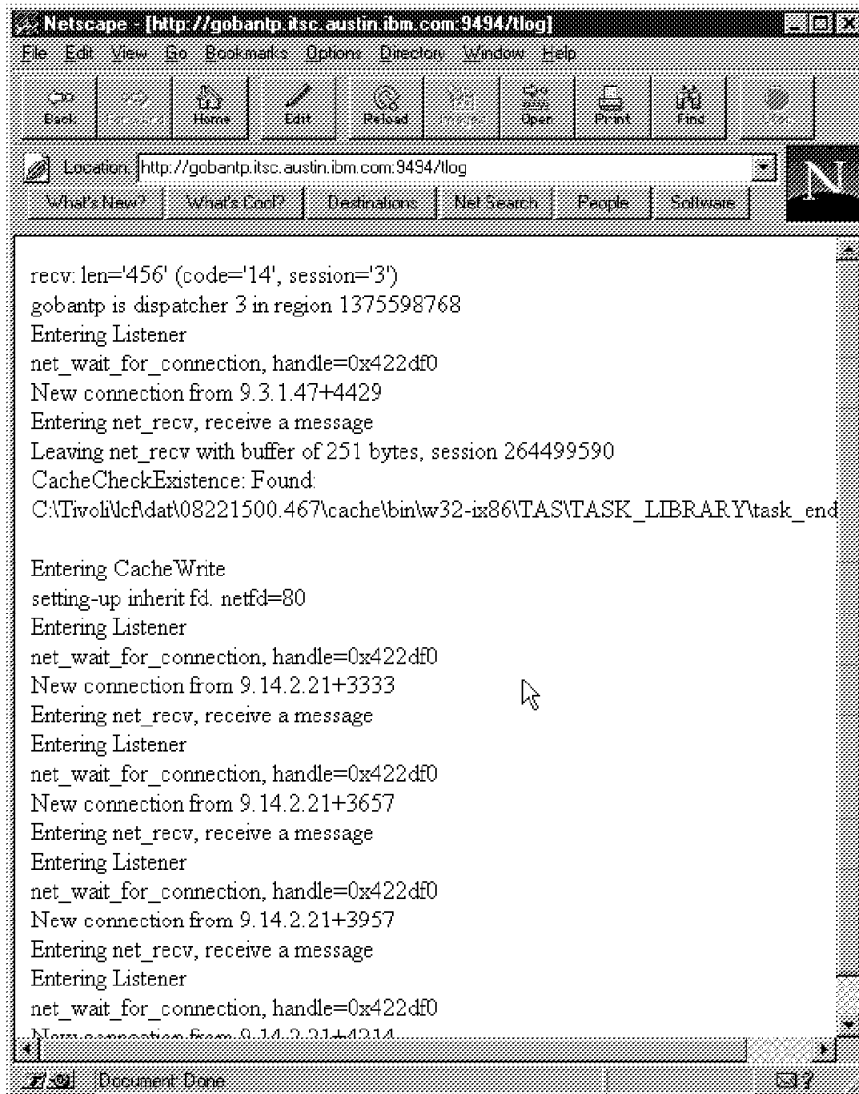


Figure 27. LCF Endpoint's Trace Log Page

### 6.2.7 Network Address Configuration Page

This Network Address Configuration Page displays information about the current Gateway settings, but more importantly, it allows you to change the configuration of an Endpoint. In the dialog presented for additional configuration options, you can specify any parameters supported by the `-D` option on the `lcfcd` command. See the *TME 10 Framework Reference Manual* for details on this command.

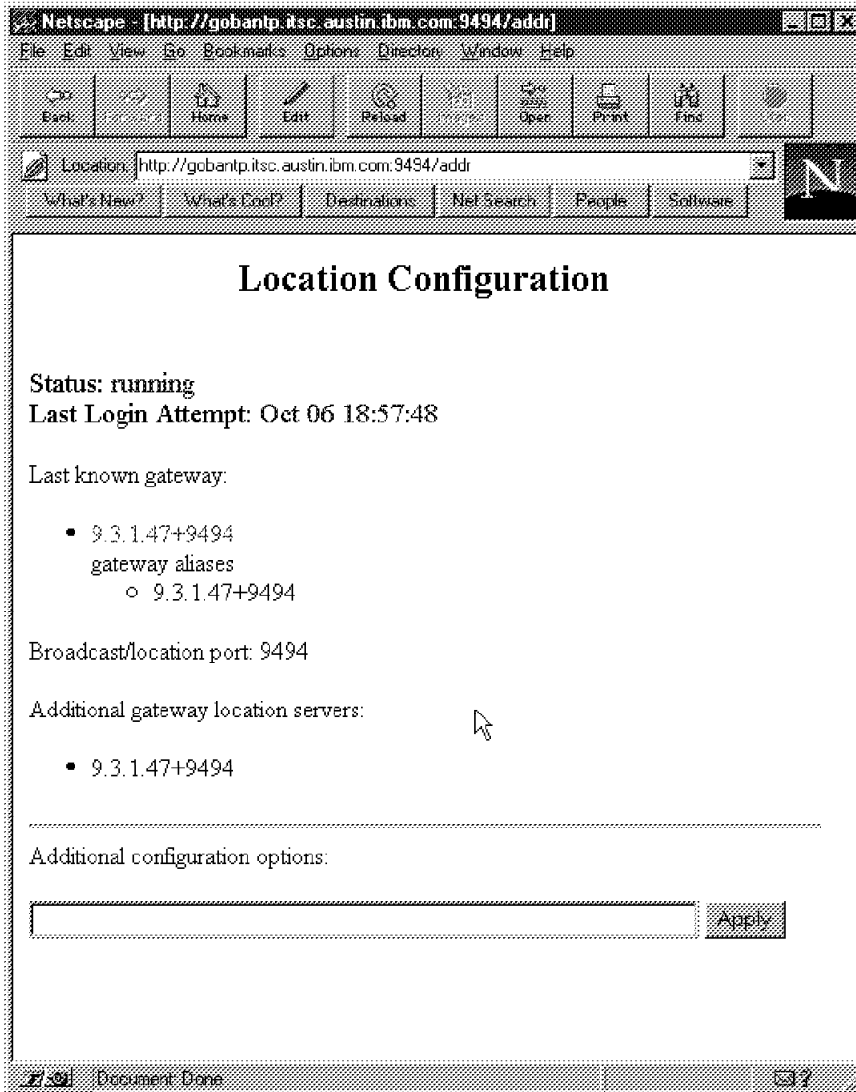


Figure 28. LCF Endpoint's Configuration Page

For instance, you could use this page to alter the default Gateway for an Endpoint. Once you have entered the configuration options and selected the **Apply** button, you will be prompted for a user ID and password. By default, the user ID is `tivoli` and the password is randomly generated. To obtain the proper password, a TME 10 Administrator can issue the `wep` command as follows:

```
wep gobantp get httpd
```

For more information on the `wep` command, refer to the *TME 10 Framework Release Notes*. In the initial release of the TME 10 Framework documentation, this command is not documented in the *TME 10 Framework Reference Manual*.

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

```
wep gobantp set http userid:password
```

To go back to changing the configuration via the 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.

---

### 6.3 Summary

This chapter has introduced a powerful new interface to the TME 10 environment. For TMR Servers, the web server function will allow applications to provide web-based interfaces to their application function. These functions can be initiated through CGI scripts or Java applets.

For LCF Endpoints, the web server function allows administrators to query information about the Endpoint and to alter the Endpoints configuration dynamically.





---

## Chapter 7. Migration from TME 3.1 to TME 3.2

This chapter provides an overview of the issues and solutions involved in migrating to V3.2 of the TME 10 Framework from V3.1. At the writing of this redbook, LCF enabled applications are not yet available, so our discussion will, by necessity, be at a high level.

However, Tivoli is committed to providing tools and facilities that will assist in the migration from the existing two-tier structure including PC Managed Nodes to the three-tier architecture provided by LCF.

One of the major factors determining how easy or difficult a migration plan will be rests on the concurrent availability of LCF enabled applications. If all applications used in your particular environment become LCF-enabled, then a migration plan can be put in place that will replace most of your current Managed Nodes and PC Managed Nodes by LCF Endpoints. Tivoli is working towards a concurrent release of LCF enabled TME 10 applications.

However, if you do have applications in your environment that do not have an LCF-enabled release concurrent with other applications, your migration plan may include the coexistence of both an LCF Endpoint and a Managed Node or PC Managed Node on the same physical system to provide support for the various applications.

---

### 7.1 Reasons for Migrating

Though we have discussed some of the benefits of LCF in previous chapters, it is valuable to include a summary of the benefits in this chapter to help clarify why you might consider migrating to LCF.

With the introduction of LCF technology, Tivoli is expanding their approach to distributed systems management. Focusing on customer requirements for a full function, lightweight client, Tivoli has addressed the following items.

- **Breaking the limit of 200 full-function clients per TMR.** In previous versions of the TME 10 Framework, the limit of 200 Managed Nodes per TMR caused customers to implement the connection of multiple TMRs solely for the purpose of supporting their large number of clients. There were, and still are, other valid reasons for implementing multiple TMRs in a single environment, but in cases where the number of clients was the only motivation, LCF will allow for a reduction in the number of TMRs. The LCF-enabled framework can support many thousand Endpoints in a single TMR.

- **Off-loading the TMR Server.** In an LCF environment with few Managed Nodes and many Endpoints, the load on the server will be reduced. For instance, in a TME 3.1 environment with 150 Managed Nodes, one server would have had to manage the TME database distributed over 150 nodes. In an LCF environment, you may have just a few Managed Nodes acting as Gateways and the majority of the systems running as LCF Endpoints. The TMR Server must manage the database distributed over only a few nodes, and much of the processing formerly done by the TMR Server is now off-loaded to the Gateway nodes.
- **TME Software Maintenance.** In a TME 3.1 environment, with many Managed Nodes, new versions of software had to be explicitly installed on all systems. The process of installing new levels of software on all systems in a concurrent fashion could be difficult in environments with a large number of clients. With LCF, new versions of applications only need to be explicitly installed on Gateway systems. When methods are invoked on Endpoints, the new versions of the methods are automatically downloaded to the Endpoint's cache.

This mechanism reduces the network overhead of down loading large numbers of files (executables) to every Managed Node concurrently. In addition, only those methods that are actually used are ever downloaded to the Endpoint. Prior to LCF, the complete set of binaries for the Managed Node piece of an application were sent to each Managed Node.

- **Improves Mass Installation Procedure.** If you have a large network, the installation of LCF Endpoints may make it easier to deploy the TME 10 environment. Using policies and configuration files as described in Chapter 4, "LCF Installation" on page 41, Endpoints can be deployed easily, without the overhead of installing a full Managed Node environment.

In addition, the broadcast technique for finding an appropriate Gateway makes the Endpoint largely immune to problems arising from relocation or reconfiguration of the managed system.

- **TME Database Consistency.** Since the TMR Database will now be distributed across fewer nodes, the process of backing up and performing consistency checks on the TME 10 database will run faster and will have fewer failure points, since it is more likely that all Managed Nodes will be active and contactable.

Although the TME 10 Framework does not maintain a database on the LCF Endpoint, individual applications could choose to write data to the local disk. In this case, any such data would be backed up through normal backup procedures for those individual workstations, not as part of the `wbkupdb` command. However, if you wish to automate this

process, it could be done through normal TME 10 Tasks/Jobs. You will need to refer to the documentation for individual applications as to whether or not they write application-specific data to an Endpoint's disk.

- **Support for New Platforms.** The spawner daemon is small and quite simple. This will make it easier to port the daemon to additional platforms. As support for additional platforms rolls out, the desire to manage these new platforms may drive the requirement to migrate to an LCF-based environment.
- **Eliminate problems associated with PC Managed Nodes.** PC Managed Nodes greatly improve the scalability and manageability of the current TME 10 architecture, but there are still a number of issues that can arise because of them:
  - Maintenance - The PC Managed Node software contains application Endpoint function. As changes are made to the applications, or new function is provided on the desktop system, the PC Managed Node code has to be upgraded. Although maintaining each node is a trivial task, multiplying it by thousands of desktop systems creates an unwanted administrative burden.
  - Security - Security for a PC Managed Node is not as strong as for platform nodes. All messages between Managed Nodes contain an encrypted authentication header to prevent a hacker from impersonating TME systems. PC Managed Nodes do not have this feature.
  - PC Managed Nodes have to be manually defined within policy regions. As in the maintenance case, this is a trivial manual task for a small number of nodes, but it becomes onerous for thousands of nodes.

---

## 7.2 Migration Process Overview

Although there are several strategies that could be employed when planning the migration to V3.2 of the TME 10 Framework, the following three steps will make up the core of your plan.

### 1. Framework Migration

LCF provides new and additional capabilities over those provided in previous versions of the TME 10 Framework. However, you can install V3.2 without initially taking advantage of LCF. In other words, you can upgrade all of the systems in your environment to V3.2, while maintaining your current two-tier architecture.

Upgrading to V3.2 uses the same procedure as installing a patch. In fact, V3.2 includes a large number of fixes to problems reported on

earlier versions of the TME 10 Framework. You should see an improvement in the performance and stability of the new Framework when installed, even when maintaining your current TMR structure.

Of course, you will want to thoroughly test V3.2 in your own test environment to ensure that all of your current applications continue to run properly.

## **2. Add New Endpoints**

As LCF-enabled applications become available, you will want to start adding new nodes as LCF Endpoints instead of Managed Nodes or PC Managed Nodes. This will be the first step in moving towards the new three-tier architecture.

During this phase, you may be defining new Profile Managers to support your new LCF Endpoints. Initially, applications will likely provide equivalent function for both traditional Managed Nodes and LCF Endpoints. However, you will need to investigate this on an application by application basis to determine whether unique profiles or profile managers will be desired to support LCF Endpoints and Managed Nodes.

At this stage of the migration, you will need to define some of Managed Nodes as Endpoint Gateways. Determining which Managed Nodes are appropriate candidates to perform the Gateway function depends on environment specific criteria, such as hardware requirements, system load, network configuration, and so on.

This phase will require a fair amount of planning and insight into the future growth of management applications and activities. During this phase you will also start adding LCF Endpoints to your existing Managed Nodes and PC Managed Nodes. This will allow a single system to be managed as either a Managed Node, PC Managed Node or LCF Endpoint. This may be a critical step if not all applications in use in your environment are LCF-enabled at the same time. (As stated before, Tivoli's goal is to have all core applications LCF-enabled concurrently.)

## **3. Migrate Managed Nodes and Profile Managers**

In the final phases of migration, you will want to convert many of your Managed Nodes and PC Managed Nodes to pure LCF Endpoints. During this phase, the subscriber lists for Profile Managers will need to be migrated such that Managed Nodes are unsubscribed, but their replacement Endpoints are subscribed. This will also require changes to the Profiles themselves to reflect the use of dataless mode.

When an LCF Endpoint subscribes to a profile, any associated data is distributed to the Endpoint Gateway. The Endpoint Gateway will update

its own database and will then manage the appropriate downcalls to the Endpoints.

The easiest way of enabling a Profile Manager to support a dataless Endpoint is to clone the ProfileManager object to a new dataless Profile Manager. It is expected that when LCF applications are released, migration facilities will be made available to migrate the information contained in current Profile Managers to new dataless Profile Managers.

---

### 7.3 Migration Scenarios

The following table summarizes the most likely upgrade scenarios for systems in current TMRs. For instance, current TMR Servers will become both a TMR Server and an Endpoint Manager. It is also possible that you will include Gateway Function on the TMR Server in some environments. Note that in many environments, fewer TMR Servers will be required and therefore, some TMR Servers will be 'downgraded' to the role of a Managed Node (and possibly Endpoint Gateway) in another TMR. Since such a step would require a complete reinstallation of the TME 10 Framework, we consider this step the same as installing a new Managed Node and Endpoint Gateway.

<i>Table 4. LCF Release 3.2</i>					
TME V3.1	TME V3.2				
	SRV-EM	MN	GW	EP	PCMN
SRV	X		X		
MN		X	X	X	
PCMN				X	X

As an example of a migration scenario, Figure 29 on page 94 depicts two interconnected TMRs that will be migrated to a single TMR as shown in Figure 30 on page 95.

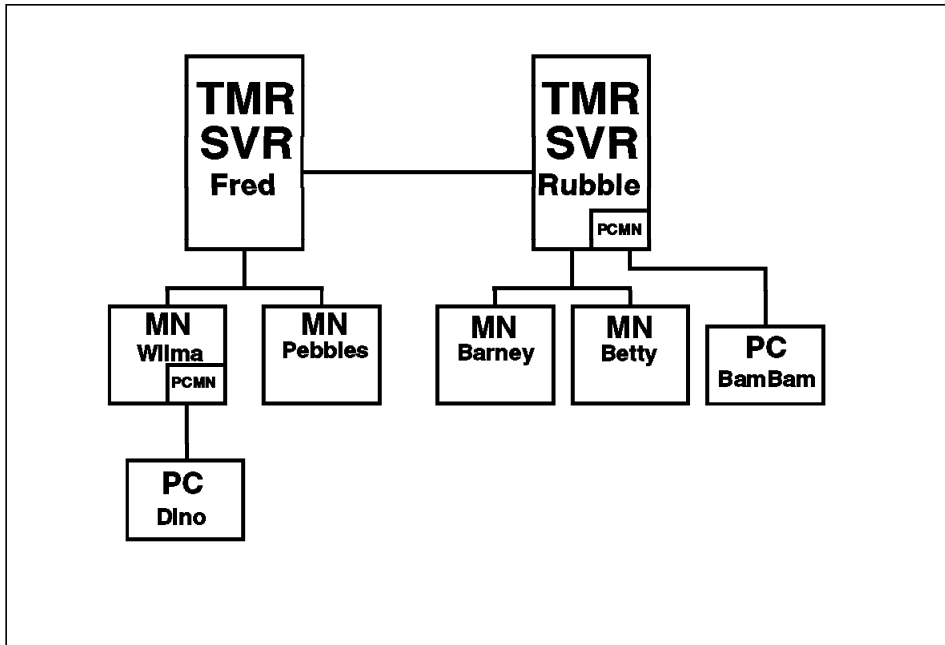


Figure 29. Two Interconnected TMRs without LCF

TMR Server Fred will continue to be a TMR Server and take on the role of Endpoint Gateway. TMR Server Rubble will no longer be needed as a TMR Server, since Fred will now be able to support many more systems as Endpoints.

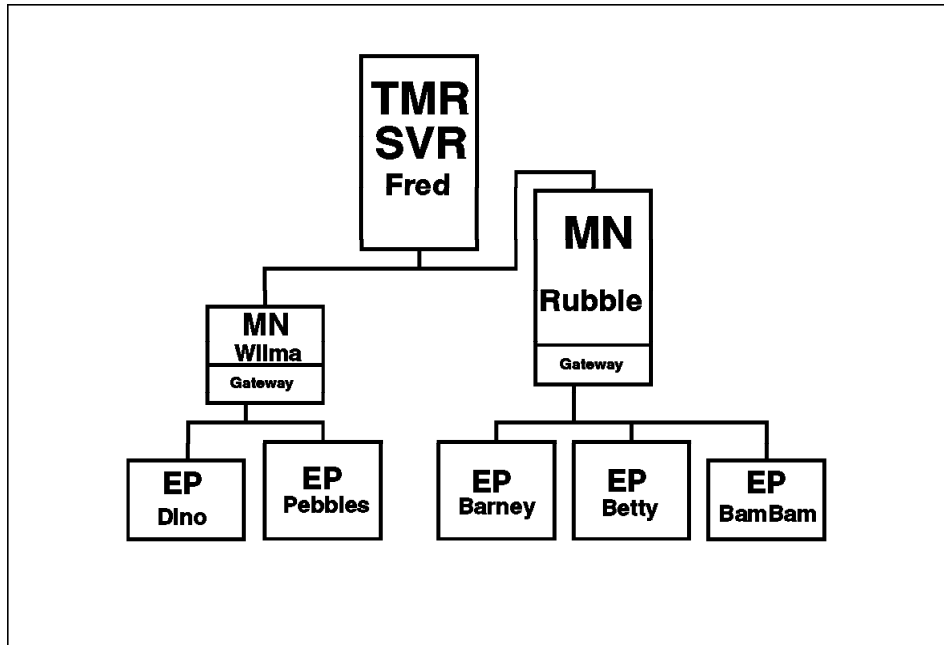


Figure 30. Single TMR with LCF

The configuration of our consolidated TMR will now have a single TMR Server, two Endpoint Gateways (Wilma and Rubble) and five Endpoints.

The first step in the migration would be to upgrade Fred to V3.2 of the TME 10 Framework. This upgrade can be installed as a patch, through the normal TME procedures. The procedure for upgrading a the TME 10 Framework will be presented later in this chapter.

The upgrade procedure will automatically make Fred an Endpoint Manager. The Managed Node Wilma could be upgraded at the same time. However, upgrading Wilma to Version 3.2 does not automatically make that node an Endpoint Gateway. That will require an additional step. However, upgrading Fred and Wilma should allow our TME 10 environment to continue running as it does today and prepare us to start migrating the other nodes.

Once Wilma is upgraded it can become a Endpoint Gateway as described in section 4.2, “Endpoint Gateways” on page 43.

The Rubble system is being migrated from a TMR Server to a Managed Node in the other TMR. There is no explicit migration mechanism for this step. Rubble must have its TME 10 software completely reinstalled. Of course, while Rubble is having its software removed and installed, the three systems in its TMR (Barney, Betty and BamBam) will not be manageable. One way

around this would be to install the LCF Endpoint software on those systems and allow them to connect via Wilma.

The Endpoint software can be installed while those systems are still running as Managed Nodes and as PC's running the PC Agent code.

Likewise Dino and Pebbles can have their Endpoint code installed while still performing their roles as a Managed Node and PC Agent.

Once Rubble is fully in place as a Endpoint Gateway, we can add Endpoint Manager policy to Fred to ensure that the Endpoints connect via the proper Endpoint Gateways.

Finally, once we have tested all of our Endpoint function and are satisfied that the Managed Nodes (other than Wilma and Rubble) and PC Managed Nodes are no longer required, we can remove the software from those systems and run our pure LCF environment.

The above scenario is obviously a simple one and many planning details are left out (such as application migration), but it provides the general steps required for most migrations.

Since most of the steps involve installing brand new software which was described in Chapter 4, "LCF Installation" on page 41, the only step we will take you through is the upgrade procedure that would be used on Fred and Wilma.

### **7.3.1.1 Upgrade Using the TME 10 Desktop**

Version 3.2 of the TME 10 Framework is available as a full product CD as well as an upgrade CD. The upgrade CD image should be used when migrating to V3.2 from prior versions.

1. From your TME 10 Desktop (see Figure 31 on page 97), select **Desktop->Install->Install Patch** and follow the instructions.



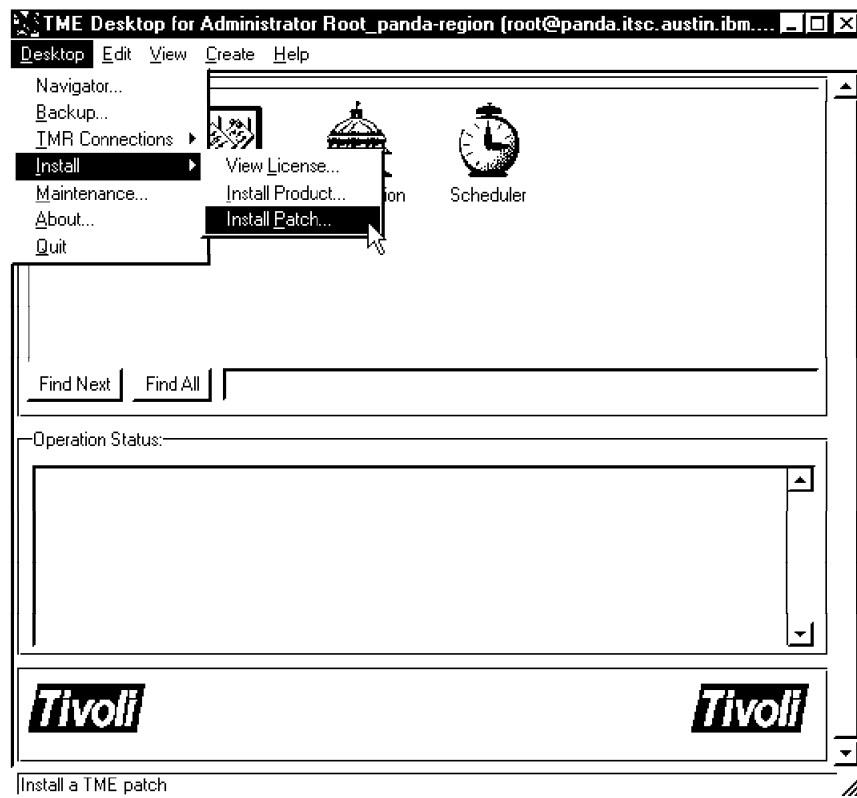


Figure 31. Upgrade to 3.2 - Start from Desktop

2. You will *not* see the error message (Figure 32 on page 98) if the path has already been selected in a previous action. In this case, continue with Figure 33 on page 99.

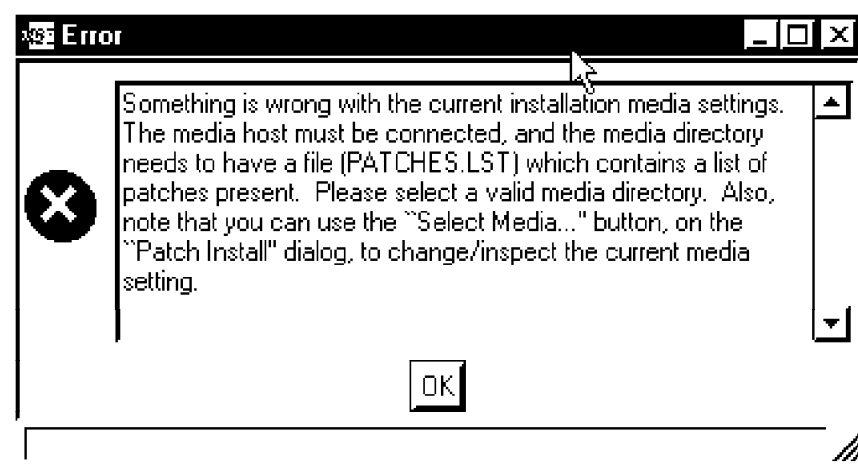


Figure 32. Upgrade to 3.2 - Install Patch - Error Message

3. Press **OK**.
4. In the dialog titled **Set Path to Tivoli Install Media**, select the host and directory on which the upgrade CD image is available.
5. In the **Install Patch** dialog (see Figure 33 on page 99), within the **Select Patch to Install** combo box, select **TME 10 Framework Upgrade to Version 3.2**.
6. Move **panda** to the **Clients to Install On** box.

At this time, you can select all Managed Nodes which are to be upgraded, as we did in this figure.

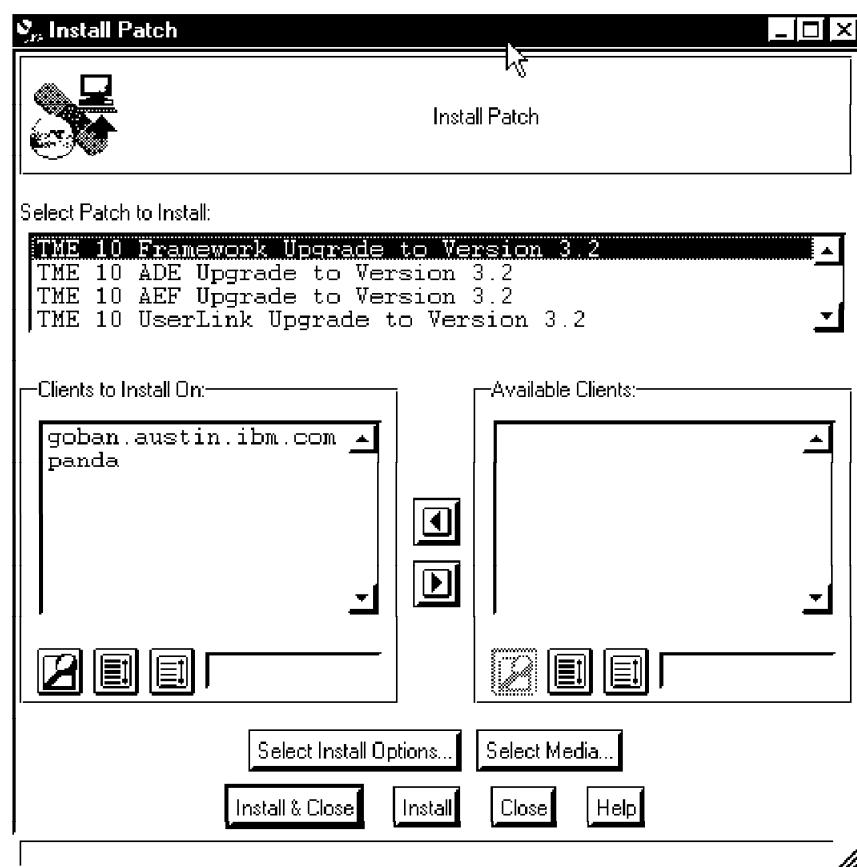


Figure 33. Upgrade to 3.2 - Install Patch

7. Continue with **Install & Close**

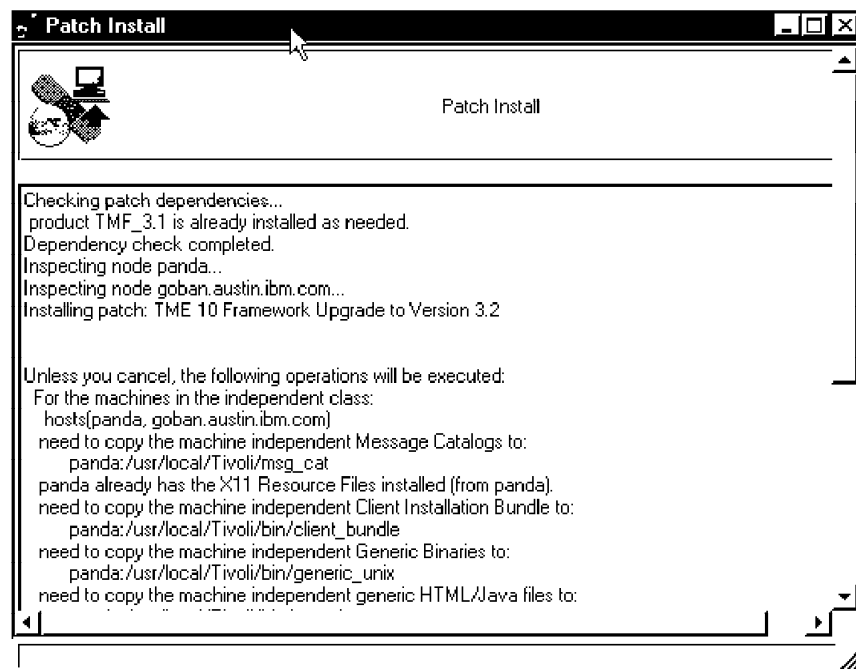


Figure 34. Upgrade to 3.2 - Patch Install

8. You will now see a series of messages displayed in the message box of the **Install Patch** panel (see Figure 35 on page 101 through Figure 37 on page 102).
9. Press **Continue Install** on the next panel (Figure 35 on page 101).

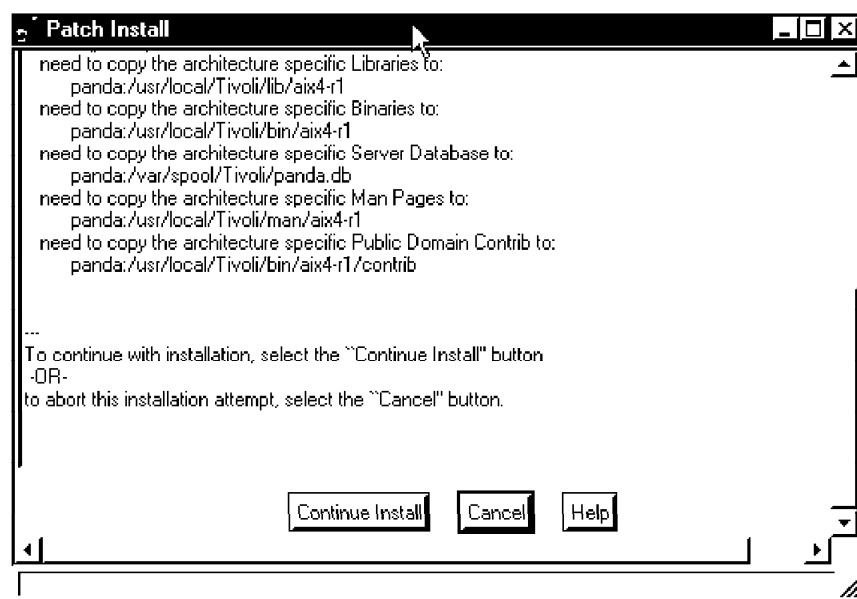


Figure 35. Upgrade to 3.2 - Patch Install (cont'd)

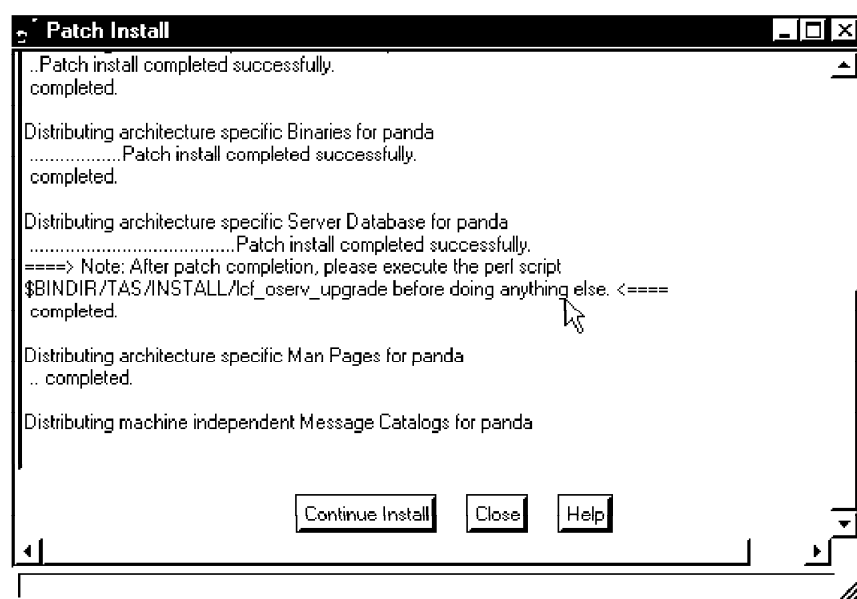


Figure 36. Upgrade to 3.2 - Patch Install (cont'd)

10. When Finished patch installation appears, select **Close** on next panel (Figure 35).

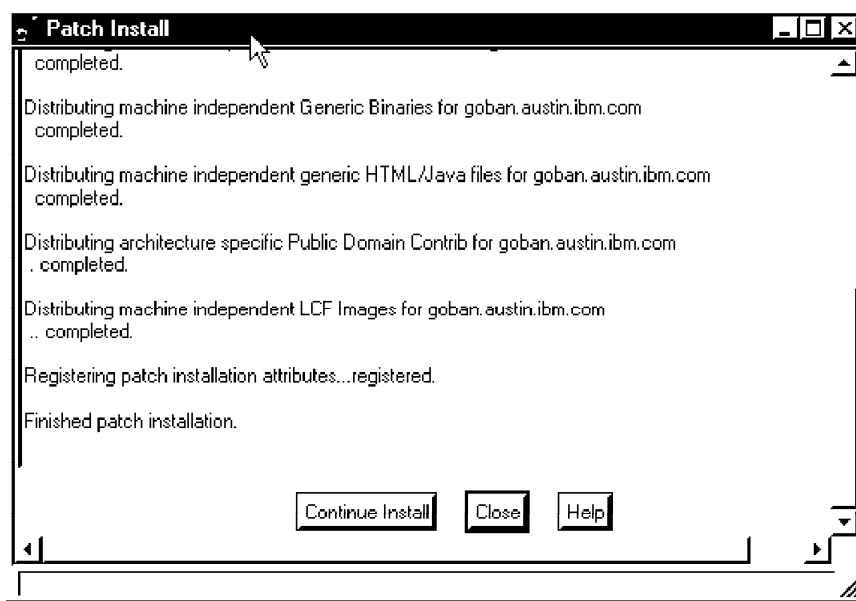


Figure 37. Upgrade to 3.2 - Patch Install (finished)

11. When the installation has completed, you'll find the new Endpoint Manager icon on your TME 10 Desktop (Figure 38 on page 103).

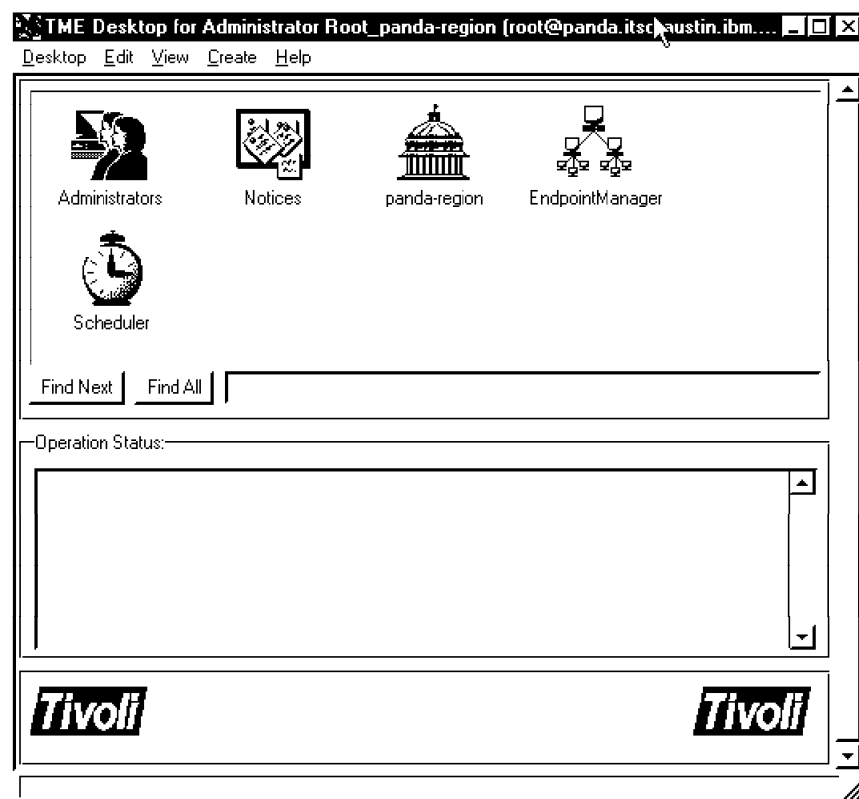


Figure 38. TME Desktop after Upgrading to 3.2

#### Important Step!

After upgrading the TMR Server, you will need to perform the following step before performing other functions.

Execute the `$BINDIR/TAS/INSTALL/1cf_oserv_upgrade` command.

This command will restart all of the upgraded oserv daemons within your TMR. If these daemons are not restarted after installing the upgrade, errors will occur.

#### 7.3.1.2 Upgrade via Command Line or Script

The TME 10 wpatch command is used to install a patch on existing systems. This command can be executed directly, but it is often embedded within a script to ensure that all environment variables and parameters are set

correctly. By upgrading a TMR Server using this procedure, the Endpoint Manager function will automatically be installed.

The following script can be used to upgrade the TMR Server to Version 3.2 of the TME 10 Framework. This script could easily be modified to upgrade Managed Nodes within the TMR at the same time. Note that the SERVER and PATCHDIR variables will need to be customized for the specific environment.

```
#!/bin/ksh!  
clear  
# Variables Definition  
SERVER=panda  
PATCHDIR=/TME32/UPDATE  
PATCHIND=TMP_32.IND  
# command line for upgrading  
wpatch -c $PATCHDIR -i $PATCHIND -y $SERVER  
$BINDIR/TAS/INSTALL/lcf_oserv_upgrade
```

See *TME 10 Framework Reference Manual* for details on the wpatch command.

Remember to backup the TMR database before and after installing the upgrade.

### 7.3.2 Add a Gateway to a Managed Node

Once Managed Nodes have been updated to V3.2 of the TME 10 Framework, Endpoint Gateway objects can be created on them. Again, this can be done via the TME 10 Desktop or through the command line interfaces.

Once the Managed Nodes (and Server) have been updated, the process is no different than shown in section 4.2, “Endpoint Gateways” on page 43.

Below is another simple example of a script that could be called to create an Endpoint Gateway.

```
#!/bin/ksh!  
#  
clear  
# Variables Definition  
GW1=traci  
#command line to create a gateway  
wrtgate -h $GW1 -p 9595  
echo done
```

Note that in this example, we have specified that port 9595 should be used for the Endpoint Gateway’s port.



---

## 7.4 Summary

This chapter has provided an overview of the various considerations involved in putting together a migration plan from previous versions of the TME 10 Framework to Version 3.2. Unfortunately, when this redbook was written, the LCF enabled applications were not yet available, so that many of the more intricate steps could not be tested and described. For instance, the use of dataless profiles and the moving of Managed Nodes to Endpoints using these profiles will likely be a process that may require a fair amount of planning. On the other hand, Tivoli is working on providing migration tools for their various applications that will handle the bulk of this for you.



---

## Chapter 8. Application Development Considerations

This chapter discusses information related to developing applications that will support LCF Endpoints. It describes the programming environment, discusses the special application library for use by LCF applications and discusses the components an LCF-enabled application should have.

---

### 8.1 Application Design

As you have read in the previous chapters, the core features of LCF include:

- Three-tier structure - Endpoint Manager, Gateway and Endpoint.
- The nature of the Endpoints - small, dataless clients that have no client database and that do not have the full TME10 Framework installed.
- The scalability possible within the LCF environment - one Gateway can manage thousands of Endpoints.

An application must be designed and written with these features in mind. The application development environment for LCF is very similar to that of the full framework. However, there are some differences in how methods are executed for Endpoints. These differences and their implications will be described in the following sections.

#### 8.1.1 TME Methods

The next two sections briefly describe how methods work in both the traditional TME 10 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 executed in the same way. From any system within the TMR, you can execute a method on any other system. The only thing you must know is the object reference. That consists of the TMR Region Number, the Dispatcher Number and the Object Number.

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 LCF Environment

Methods run differently in the LCF environment than in the full framework environment. The differences are:

1. An application running on an 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. It does require that an application that will require Endpoints to manipulate remote objects to have an Endpoint Gateway component. The Endpoint will then talk to this Endpoint Gateway component which will in turn take advantage of the fact it has full access to the TMR to invoke the appropriate methods on the target object(s). This design enables a higher degree of scalability than was possible in previous versions of the TME 10 Framework.
2. In the LCF, method calls differ according to whether they are upcalls (methods invoked from an LCF Endpoint) or downcalls (methods targeted for an object residing on an Endpoint).
3. Endpoint methods use some of the tools and services of the TME10 ADE, but they are implemented using a special application mini-runtime library (**libmrt**) specific to LCF.

The above differences introduce some new terms, such as:

- **LCF Object** An object running on an Endpoint.
- **Endpoint method** A method that runs on an Endpoint. An Endpoint method runs as the result of a downcall.
- **Gateway method** A method that runs on an Endpoint Gateway, usually as the result of an upcall made by the Endpoint associated with the Gateway.
- **Downcall** A method invocation from the Gateway "down" to an Endpoint.
- **Upcall** A method invocation from an Endpoint "up" to the Gateway.

### 8.1.1.3 LCF Object

An LCF object is an object running on an Endpoint. LCF object references have the special form:

**R.D.P+** where:

- **R** is the TMR number. Objects in TME V3.x have 10 digit region numbers. Objects in Versions 2.x have six digit region numbers.
- **D** is the number of the object dispatcher or host number within the TMR.

- **P** is the object number of the prototype object of the class for which the method is defined.
- **+** indicates an LCF object reference.

This combination of the three numbers always makes the object reference unique. On the Endpoint there is no need to create an instance of each object as you do in the full framework. LCF objects do not have a unique per-object state maintained for them by the object system. Thus, there is no need to notify the object system when a new Endpoint object is created or deleted.

To create a new object based on an existing prototype object, you generate a reference to it in the above form and then invoke it. After that, 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 and maintained by the framework, applications have to assume responsibility for maintaining their own persistent store outside the context of the LCF base services.

To find the class of an object you can issue the command:

```
objcall <R.D.P> getattr class_objid
```

The following example shows this command and its output:

```
[root@panda]/u> objcall 1648772799.1.343 getattr class_objid
d03648772799.1.18#TMF_SysAdmin::InstanceManager#
[root@panda]/u>
```

#### 8.1.1.4 Endpoint Methods

An Endpoint method is a method that runs directly on an Endpoint. It is implemented using the special application mini-runtime library, **libmrt**, which provides a subset of the TME10 operations.

When an Endpoint method is invoked on an object on the 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 these ways:

- Endpoint methods must be single threaded. There are no daemons or multiple-entry points for Endpoint methods.
- There is no transaction support for Endpoints in the LCF environment.
- You must specify dependencies for Endpoint methods. An Endpoint method may be implemented across several files. Besides the

executable containing the method, a 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 Gateway will ensure that all dependencies for a method have been downloaded to the Endpoint before invoking a 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 Endpoints are initially installed. Method bodies are identified in the standard method header and are downloaded when needed. Because the dependencies (the supporting files) must also be downloaded when missing, the dependencies must be called out in the Endpoint method so they can be present when needed.

#### **8.1.1.5 Gateway Methods**

A Gateway method runs on the Endpoint's Gateway. It runs as the result of an upcall request from an Endpoint or as the result of a request from another Managed Node. Since the Endpoint can not communicate directly with other nodes, it must initiate all actions by invoking an upcall on a Gateway method. The Gateway method then takes advantage of the full TME 10 Framework to resolve and locate the appropriate target object(s) and invoke the proper methods.

#### **8.1.1.6 Downcalls and Upcalls**

A process called a downcall is when a method request originates on a Managed Node and executes on an Endpoint. The Endpoint receives the program name to execute and the 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 is on the Endpoint but out of date, the method executable is downloaded to the Endpoint. If the method has dependencies, they are also downloaded.

An upcall occurs when Endpoint applications initiate TME 10 operations elsewhere within the local TMR. The Endpoint can only invoke methods on the Managed Node associated with its Gateway, not on any arbitrary object in the TMR. This design maintains scalability since upcalls are handled at the Gateway without going to the TMR Server each time.

The upcall consists of a class name, the name of the method to be run and the arguments for the method. The Gateway then resolves the prototype object for the class name and constructs the object call to invoke the method. Not all Endpoint applications need to make upcalls. An application that supports upcalls must provide both the Endpoint code and the Gateway upcall method.

**Upcall Architecture:** If an LCF-enabled application will require an Endpoint to make upcalls, then an instance of an upcall collector will be required. An upcall collector is a daemon that implements all Gateway methods for an application, and executes in the Gateway's Managed Node. It is basically a store-and-forward router for application-specific upcalls. The upcall collector collects upcall data, optionally batches them together and forwards the data to an application Mid-Level Manager (MLM). The application MLM then processes the data and, depending on the application, possibly forwards it to a Top-Level Manager (TLM on the Server) for the application, or invokes methods on other objects directly. Since the Gateway resides on a full function Managed Node, the mid-level manager has full access to the framework services.

The upcall collector is implemented by an abstract class object. An abstract class provides a base class from which you can derive other classes. It can be used for inheritance purposes. Abstract classes frequently describe the characteristics of the other classes and instantiable classes describe the characteristics of individual objects. There is exactly one abstract object for each class on each object dispatcher. Because the upcall collector is implemented by an abstract object, it has no persistent store. There is no client database installation for an LCF application and so there are no client database updates or any related issues concerning maintaining the consistency of the database.

The upcall collector component is defined as follows:

- The upcall collector has its own IDL interface.
- Upcall collectors will never be instantiated, so they are defined to be abstract implementations. That is, the TEIDL keyword **abstract** appears in the implementation header in the **.imp** file.
- Dependencies for upcall collector methods are specified using the **wdepset** and **wchdep** commands.

For Endpoint methods that will be making upcalls, there are two ways of handling authorization:

- Short-lived methods that need to perform only one upcall can use per-upcall authorization. In this case, the upcall and principal login are combined into a single step to authorize a single upcall.
- Longer-running Endpoint methods that require multiple upcalls use a principal login. The principal login returns credentials that may be used to authenticate subsequent upcalls. The Gateway authenticates the user name and password, then returns a "sealed certificate". The application should store the certificate on disk or another persistent store where it may be used by others.

### 8.1.2 Components of an LCF Application

When designing your LCF application, keep in mind that there is no TME 10 database on Endpoints. Therefore, objects on Endpoints do not have a persistent state.

Endpoint applications that need to perform TME operations must be written with both a client (Endpoint) and a server component. Depending on the application, you may also need to write an optional Gateway component. A list of the possible components is:

- Client component (Required) - Includes methods that run on the Endpoint.
- Upcall collector (Optional) - Required if your application makes upcalls.
- Mid-Level Manager (Optional) - Required if the Upcall Collector is present.
- Top-Level Manager (Optional) - Related to the Mid-Level Manager and may be used to keep configuration state information for all Endpoints for example.
- Gateway component (Optional) - Present only if the application makes upcalls.
- Server component (Required) - Runs on the TMR server.

In the following sections we present three scenarios to illustrate the Endpoint, Gateway and Server components of an LCF-enabled application.

#### 8.1.2.1 Downcall Example

To illustrate a downcall example, consider a scenario related to an implementation of the TME 10 Software Distribution push interface. In this case the application developer writes two modules:

- Server module, which distributes files to the Endpoint.



- Endpoint module, which receives, unpackages and installs the files sent from the server.

The flow of this application includes:

1. The Server initiates a file package distribution to an Endpoint.
2. The file package is distributed through the Gateway. There is no application specific code required at the Gateway. The Gateway will receive such methods, verify that the Endpoint cache contains the required method and its dependencies, and then invoke the method on the Endpoint itself.
3. The Endpoint method unpackages and installs the file package.

#### **8.1.2.2 Upcall Example**

To illustrate an upcall example, consider the scenario related to an implementation of the TME 10 Software Distribution pull interface. In this case the application developer writes three modules:

- Endpoint module initiates a file or package pull and unpackages and installs the files.
- Gateway module handles the pull request from the Endpoint and using the full function TME library, initiates a request that is sent on to the server.
- Server module receives the request and distributes the file packages to the Endpoint.

The application flow is as follows:

1. The Endpoint initiates a file package distribution. This request is sent to the Gateway as a method call on the Gateway module.
2. The Gateway module interprets this request and calls the appropriate methods on the application server.
3. The server receives and responds to this request and distributes the file packages to the client, as in the pull scenario above.
4. The Gateway interprets the method call, ensures that the appropriate receiver methods are in place on the Endpoint, and then invokes those methods.
5. The Endpoint unpackages and installs the file package.

#### **8.1.2.3 Upcall Collector Example**

To illustrate an example involving an Upcall Collector, we could envision an inventory application designed to collect inventory information from a

machine each time it reboots. As part of this, an initialization method has been run on each Endpoint to run the inventory program on boot.

This application requires four modules:

- Endpoint module, which queries the system and generates an inventory report.
- Gateway module, which gathers the data from multiple Endpoints.
- Upcall Collector module, that collects information from various Endpoints before forwarding the information on to the Gateway module.
- Server module, which processes the various inventory reports and adds the information to a relational database.

Logically when a boot occurs, the following steps are taken:

1. The Endpoint runs a program that generates a local inventory report
2. The Endpoint program then invokes an "inventory report" method on the Gateway and passes the inventory data as a parameter.
3. The Upcall Collector waits until it has responses from some predetermined number of Endpoints.
4. The Gateway invokes a method on the TMR server to pass the collected inventory reports to the server,
5. The server component then processes the collected information and updates a relational database.

---

## 8.2 The LCF programming Environment

Programmers use the Tivoli Extended IDL (TEIDL) compiler to produce method stubs and skeletons. The stubs are used by TME methods to invoke Endpoints methods. The skeletons are linked with Endpoints methods and the LCF application runtime library to create executables.

LCF methods are described using four standard files with the following extensions:

- **.idl** - These files contain the definition of the interface for one or more classes of objects.
- **.imp** - These files contain information about the implementation of the interfaces described in IDL files.
- **.prog** - These files specify execution characteristics for methods in a class.

- **.ist** - These files define how classes are installed in a TME 10 environment.

All of these files are used to build the methods on the server, the Gateway, and the Endpoint. The only difference is that for the Server and Gateway methods you use the standard Tivoli Extended IDL (TEIDL) compiler and for Endpoint methods you use a special front end script (`ltid`) to run the TEIDL compiler. The `ltid` processes the standard TEIDL compiler-generated code to enable it to run on an Endpoint. At link time the following libraries are used:

- **libmrt** - The application mini-runtime library for LCF Endpoints.
- **libcpl** - The common porting layer library.
- **libdes** - The DES library.

In addition, Endpoint methods must be linked to any other libraries needed by the specific methods.

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` option when starting `lcfg`. With this option you can stop each Endpoint method when it starts and attach a debugger. In this way you can debug all methods on an Endpoint. For example, to debug you can use the command:

```
lcfg -D debug_flags=1
```

When each method start, `lcfg` prints to the console the method name and the process it starts. It then suspends the methods so you can attach a debugger.

---

### 8.3 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 an LCF method executable for an Endpoint. Using this special application runtime library helps keep the Endpoint portion of the application as small and simple as possible, while still providing the capability to develop robust and complete applications. **libmrt** contains subsets of the following sets of functions provided by the full framework:

- Memory management
- Distributed exceptions

- Sequence manipulations
- File system input/output
- Logging functions
- ADR marshalling functions

In general, the functions in **libmrt** are part of the full framework and function identically to the full framework functions (look for these functions in the *Tivoli Application Services Manual, Volumes 1 and 2*).

The following sections summarize the functions available in the **libmrt** library.

### 8.3.1 Memory Management

The library provides a subset of memory management functions that work with either local or global memory:

- Use local memory for temporary allocations that are automatically freed when they go out of scope.

- Use global memory for all allocations that must persist.

The following table summarizes the memory management functions available for Endpoint methods.

<i>Table 5 (Page 1 of 2). Memory Management Functions</i>	
<b>Routine Name</b>	<b>Description</b>
<b>mg_malloc</b>	Allocates a block of uninitialized global heap memory
<b>mg_free</b>	Free global memory allocated by <b>mg_malloc</b> , <b>mg_calloc</b> or <b>mg_realloc</b>
<b>mg_calloc</b>	Allocates a block of global memory, initialized a zero
<b>mg_realloc</b>	Reallocates global memory; changes the size of an allocated block of memory
<b>mg_strdup</b>	Copies a string into a new block of memory allocated with <b>mg_malloc</b>
<b>mg_cleanup</b>	Free all globally allocated memory ( <b>mg_*alloc</b> ) that has not been deallocated
<b>ml_create</b>	Creates a new local memory heap
<b>ml_malloc</b>	Allocates uninitialized local memory
<b>ml_free</b>	Frees local memory allocated with one of the <b>ml_*alloc</b> functions
<b>ml_calloc</b>	Allocates a local memory, initialized to zero
<b>ml_realloc</b>	Reallocates local memory

<i>Table 5 (Page 2 of 2). Memory Management Functions</i>	
<b>Routine Name</b>	<b>Description</b>
<b>ml_to_mg</b>	Moves memory from local to global
<b>ml_strdup</b>	Duplicates string using <b>ml_malloc</b>
<b>ml_destroy</b>	Frees all memory in a local memory heap
<b>ml_ex_malloc</b>	Allocates uninitialized local memory, in a Try() frame
<b>ml_ex_calloc</b>	Allocates local memory, initialized to zero, in a Try() frame
<b>ml_ex_realloc</b>	Reallocates local memory, in a Try() frame
<b>ml_ex_strdup</b>	Duplicates string using <b>ml_ex_malloc</b> , in a Try() frame

### 8.3.2 Distributed Exceptions

Exceptions are used to report fatal errors. You can use the standard **try/Catch** frame macros or the variable argument exception functions to handle exceptions.

The following table describes the macros you can use.

<i>Table 6. Distributed Exceptions - Available Macros</i>	
<b>Macro</b>	<b>Description</b>
<b>Try</b>	Starts a new Try/Catch frame
<b>Catch</b>	Catches an exceptions of a given type
<b>CatchAll</b>	Catches exceptions of any type
<b>EndTry</b>	Ends a Try/Catch frame
<b>Throw</b>	Throws an exception
<b>ReThrow</b>	Rethrows a caught exception
<b>ev_to_exception</b>	Converts an environment to an exception
<b>exception_to_ev</b>	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 you can use.

<i>Table 7. Distributed Exceptions - Available Variable Argument Exception Functions</i>	
<b>Routine</b>	<b>Description</b>
<b>vaThrow</b>	Throws an error message
<b>vaThrowDerived</b>	Throws a type of error message
<b>vaMakeException</b>	Returns a pointer to the exception
<b>vaAddMsg</b>	Appends a new message to an X/Open message
<b>ThrowExErrorMsg</b>	Throws a message as an exception

### 8.3.3 Sequence Manipulations

The data types supported by IDL do not include variable length arrays. Instead, you must use a data type called a sequence. A sequence consists of a pointer to an array of a given data type and a count of the number of elements in the array.

The LCF environment supports a limited subset of the functions available to manipulate sequences, and their use is slightly different from that used in the full framework. In the full framework, the sequence APIs are all implemented as function calls in **libtas**. You must type cast all references of user-defined sequences from the native data type to the **sequence\_t** type. In LCF, the sequence APIs are lightweight macros defined in **seq.h**. In LCF, it is no longer necessary to cast from the user-defined type to the **sequence\_t** type. All LCF sequence macros allow you to use the sequence without having to cast values to and from **sequence\_t**. The following table shows the supported sequence macros, where *<type>* is a typed **sequence\_t**.

<i>Table 8 (Page 1 of 2). Sequence Manipulations - Available Sequences Macros</i>	
<b>Macros</b>	<b>Description</b>
<b>Seq_new( size_t size)</b>	Creates a sequence of elements of the size specified in memory
<b>Seq_zero( &lt;type&gt; *seq)</b>	Clears the sequence
<b>Seq_len( &lt;type&gt; *seq)</b>	Returns the number of elements in the sequence specified
<b>Seq_get( &lt;type&gt; *seq, int index)</b>	Returns a pointer to the data item in the sequence for the index specified
<b>Seq_add( &lt;type&gt; *seq, &lt;type&gt; item)</b>	Adds a data item to the end of a sequence
<b>Seq_remove( &lt;type&gt; *seq, int index)</b>	Removes a data item from a sequence

<i>Table 8 (Page 2 of 2). Sequence Manipulations - Available Sequences Macros</i>	
<b>Macros</b>	<b>Description</b>
<b>Seq_free_buffer</b> ( < type > *seq)	Frees the memory allocated for the buffer portion of the sequence specified

### 8.3.4 File System Input/Output

LCF supports a set of functions for file system input and output, which are summarized in the following table.

<i>Table 9. File system input/output - Available Functions</i>	
<b>Function</b>	<b>Description</b>
<b>open_ex</b>	Opens a file. Throws an exception on error.
<b>read_ex</b>	Reads a file. Throws an exception on error.
<b>write_ex</b>	Writes a file. Throws an exception on error.
<b>close_ex</b>	Closes a file. Throws an exception on error.
<b>makedir_ex</b>	Makes a directory. Throws an exception on error.
<b>make_path</b>	Checks for and build every component of the path.
<b>does_file_exist</b>	Returns true or false.
<b>get_file_length_ex</b>	Returns the number of bytes in a file.
<b>get_open_file_length_ex</b>	Returns (in bytes) the size of a file opened with <b>open_ex</b> .
<b>lseek_ex</b>	Moves around in a file opened with <b>open_ex</b> .
<b>rename_file_ex</b>	Renames a file. Throws an exception on error.
<b>copy_file_ex</b>	Copies source path to destination path and returns the number of bytes copied. Throws an exception on error.
<b>ep_stream_read</b>	Reads a <b>mdist</b> stream.

### 8.3.5 Logging Functions

The logging utility provides functions that enable you to create multiple logs to produce **printf**-style messages to a console and to a file. The logging utility includes the following functions.

<i>Table 10. Logging Functions - Available functions</i>	
<b>Function</b>	<b>Description</b>
<b>LogInit</b>	Creates a new log file. It also backs up the old log file and allocates resources needed by the log module.
<b>LogDeinit</b>	Deallocates resources set by a call to <b>LogInit</b> .
<b>LogMsg</b>	Uses the Tivoli National Language Support (NLS) to format an internationalized message and then output it to console and log file.
<b>LogSetDefault</b>	Maintains a static (private) pointer to the default log structure.
<b>LogGetDefault</b>	Returns a pointer to the default log structure.
<b>LogSetThreshold</b>	Sets the output threshold of the display level of the requested log.
<b>LogGetThreshold</b>	Returns the value of the display_threshold for that log.
<b>LogSetOutputStdout</b>	Sets the boolean to output messages to stdout.
<b>LogGetOutputStdout</b>	Returns the values of the output_stdout for that log.
<b>LogSetAppName</b>	Sets the identifier to be used in logging messages.
<b>LogGetAppName</b>	Returns the value of the application name for the requested log.
<b>LogQ</b>	Is a wrapped function around <b>LogMsg</b> ; it implements a circular queue in memory of the last n messages, to be included in exceptions.
<b>LogQueueAlloc</b>	Allocates the size of the buffer <b>LogQ</b> messages.
<b>LogQueueDealloc</b>	Shuts down and deallocates memory used for <b>LogQ</b> buffer.
<b>LogQueueGetSize</b>	Queries the size of the buffer for <b>LogQ</b> messages.
<b>LogQGetBuffer</b>	Returns a character array containing the circular queue of <b>LogQ</b> messages.
<b>LogData</b>	Formats and logs binary data.

### 8.3.6 ADR Marshalling Functions

The abstract data representation (ADR) functions are identical to the full framework version with one exception: there is no interface repository on LCF Endpoints. Because there is no dynamic type lookup, all types used 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, it is possible for a program to 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 a common place to store additional information associated with interfaces to ORB objects.

The following data types are built in to **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 The Common Porting Layer Runtime Library

The Common Porting Layer library (**libcpl**) provides functions that either exist on some but not all platforms or that exist but behave differently on a particular platform. The **libcpl** includes the following functions:

- **Binary tree search functions** - These routines are for manipulating binary search trees. All comparisons are done with a user-supplied routine. This routine is called with two arguments: pointers to the elements being compared. It returns an integer less than, equal to, or greater than zero, according to whether the first argument is to be considered less than, equal to or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared. These functions are:
  1. **tsearch(...)** is used to build and access the tree. If there is a datum in the tree equal to the value pointed to by the argument , a pointer to this found datum is returned. Otherwise, the argument is

inserted, and a pointer to it returned. Only pointers are copied, so the calling routine must store the data.

2. **tfind(...)** will search for a datum in the tree, returning a pointer to it if found. However, if it is not found, **tfind(...)** will return a NULL pointer.
3. **tdelete(...)** deletes a node from a binary search tree. **tdelete(...)** returns a pointer to the parent of the deleted node, or a NULL pointer if the node is not found.
4. **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.

- **cpl\_correct\_path** - This function corrects the path by replacing slashes with the appropriate path delimiter for the particular operating system.
- **Directory entry functions** - These functions enable the caller to use these APIs for all platforms without having to be concerned with platform specific implementation details. The functions are:
  1. **cpl\_opendir(...)**
  2. **cpl\_rewinddir(...)**
  3. **cpl\_closedir(...)**
  4. **cpl\_telldir(...)**
  5. **cpl\_seekdir(...)**
- **get() functions** - These are a variety of UNIX system calls that are not provided on other platforms. A brief description of these functions follows.
  1. **cpl\_getcwd** - Returns the current working directory of the calling process.
  2. **cpl\_getenv** and **cpl\_putenv** - Gets and puts environment variables.
  3. **cpl\_getopt** - Extracts the command line switches and their arguments from the command line.
  4. **cpl\_getpass** - Queries user for a password (string) from the standard input. The characters entered by the user are not echoed.
  5. **cpl\_gettimeofday** - Returns the time zone.
  6. **cpl\_gethostname** - Returns the hostname of the system.

- **ltoa** converts a long integer from a binary representation to a string representation.
- **stat** Macros. The **stat** function is often accompanied by a series of macros that provide a simple interface to obtain useful information about files. The MSVC runtime does not provide an implementation for these macros so they are defined in the Common layer. The following macros are defined:
  1. **S\_ISDIR(...)** - Is the file a directory?
  2. **S\_ISCHR(...)** - Is the file a character special device?
  3. **S\_ISFIFO(...)** - Is the file a FIFO?
  4. **S\_ISREG(...)** - Is the file a regular file?
- **printf, fclose, fopen, getc** - Because these functions may not exist on the Windows and Netware platforms, the Common layer furnished wrappers for these. They are:
  1. **cpl\_printf**
  2. **cpl\_fclose**
  3. **cpl\_fopen**
  4. **cpl\_getc**
- **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 (for example, in the MSVC compiler has some limitations on the temporary directories). These APIs are:
  1. **tmpfile()** - The **tmpfile** function creates a temporary file and returns a pointer to that stream (like ANSI/C function).
  2. **tmpnam(...)** - The **tmpnam** function generates a temporary filename that can be used to open a temporary file without overwriting an existing file (like ANSI/C function).
  3. **tmpdir(...)** - It is an additional function, it enables the application to query for the location of the system temporary directory on a specific platform.
- **Thread yield functions** - This function, **cpl\_THREADyield**, yields a timeslice for Windows, Windows 95 or Netware platforms.
- **Miscellaneous functions** - There are other scattered functions that are also included in the Common Porting Layer.

---

## 8.5 Summary

This chapter has provided an overview of the application development model that must be employed in an LCF environment. In addition, we have shown most of the functions that are included in the mini-runtime library (**libmrt**) that is available to applications running on an LCF Endpoint.

---

## Chapter 9. Useful Tips and Scripts

This chapter discusses some general hints and tips about working with TME 10 Framework V3.2. It also provides examples of some useful scripts that you may want to customize and use in your environment. The information and sample scripts shown include various tips and shortcuts we found while creating this redbook.

---

### 9.1 Backup and Recovery

In this section we include some general information about backing up and recovering your TME 10 database. The TME 10 database is critical to the proper operation of your TME 10 environment and applications and so it is important that you put procedures in place to ensure that backups are created with appropriate frequency.

#### 9.1.1 Backup Process

This section describes the process that the TME 10 Framework uses to backup its database.

At install time, a backup object is created for each Managed Node. You can see the references to these objects using the following command:

```
#wls -o /Library/BackupClient  
200200.1.342#TMF_Backup::Client# panda  
200200.2.13#TMF_Backup::Client# fred  
200200.3.13#TMF_Backup::Client# traci
```

When a backup is initiated, the following steps are taken:

1. The backup file is created and opened.
2. The backup objects for each of the clients are contacted (one by one) and directed to begin a backup.
3. Each managed node synchronizes its database to write any outstanding transactions.
4. Clients tar and compress the necessary files in their database directory using a snapshot method.
5. Data is then transferred back to the Server using the standard TME communications mechanisms.
6. The TMR Server backs up its own database, and then combines its own backup and each of the client's backups into a single file.

When you issue a backup, the Server compares its data with the clients' data, and clients are only requested to make their own backups if the Server considers it necessary.

7. Notices regarding the backup are logged and sent to a notice group.

By default, the file containing the backup is stored in `$DBDIR/./backups`. The file name format is

```
DB_<Date>-<Time>
```

It is suggested that you rename this file to add some description at the end of the file so that the proper level of the database can be restored when needing to recover or back off changes that have been made.

For example:

```
# mv DB_Oct21-1200 DB_Oct_21-1200.Bedrock.b4.Sentry.installation
```

For more detailed information about the `wkupdb` command, refer to *TME 10 Framework Reference Manual*.

If you issue an `odstat -c` during a backup process, you will see that backups are serial. In other words, the Server makes calls on a node by node basis. To perform a complete TMR database backup with a large number of Managed Nodes can take quite some time, depending on the sizes of the various databases. One of the benefits of LCF is that it will reduce the number of Managed Nodes (by replacing Managed Nodes with Endpoints). Therefore, the backup process will only be serialized across a few systems, rather than on each system (full Managed Node) in the TMR, resulting in much less elapsed time for a TMR backup.

### 9.1.2 Common Backup Issues

Some of the common backup related items that frequently raise questions from new administrators include:

- Notices are not restored if you perform a normal TME10 backup.  
Typically, one would not want to restore the notice set back to a previous level (losing any notices that had arrived since the backup was taken). Therefore, notices will not be restored back to the state they were in at backup time.
- There are two options that can be used during restore. The use of these options sometimes causes confusion.
  - `-r` option causes the `oserv` to be restarted after the restore. Thus, the restored database is immediately made the active database.

- **-r -R** copies the backup files to a directory without restarting the oserv. Therefore, the oserv continues to run with the current database. The next time the oserv is restarted, it will utilize the restored database.
- The default directory does not get changed if the database directory changes. Use the following commands to look up or change the default backup directory:

```
BO=wlookup TMRBackup
idcall $BO _get_default_device
"/usr/Tivoli/overlook/202/db/backups/DB_%t"
idcall $BO _set_default_device "/usr/Tivoli/backups/DB_%t"
```

- Cause of errors when backing up the database. This is often caused by a partial installation that was not recognized as having not completed successfully. To identify this particular case, use the following commands:

```
wlookup -ar ManagedNode -n <managed_node_name>
```

A fully installed managed node looks like:

```
TaskExecute      1081780928.6.9#TMF_ManagedNode::TaskExecute#
BackupClient     1081780928.6.13#TMF_Backup::Client#
imp_TMF_UI::DesktopList 1081780928.6.15#TMF_UI::DesktopList#
imp_TMF_UI::ExtD_DesktopList 1081780928.6.28#TMF_UI::ExtD_DesktopList#
```

If you are missing any of these entries there is a possibility that the client was not fully installed and will need to be removed and re-installed.

- Cause of database busy message.  
If you get an error message indicating a busy database, this typically comes from a snapshot and indicates that the database is being updated too frequently to be able to tar the data before it changes. This is caused by the oserv process on the machine being too busy to capture a clean snapshot of the database. For example, a client with Sentry monitors scheduled for very short intervals might give such message.
- If alternate IP addresses are used between the Server and Client you might get a cannot map IP message. This error is in the Servers' oservlog. Generally, we can say that *arp* and *rarp* must show the correct information. Routing paths should be checked as well, or you may get an IOM time-out error.
- Cause of directory permission errors. When backups are performed, the system creates the backup file with the permission of the administrator running the backup. Errors can occur during backup because the administrator does not have the correct permissions. Several ways of bypassing this inconvenience include creating a task that runs as **root** that performs the `wbkupdb` command.

### 9.1.3 Run wchkdb Before Any Backup

The wchkdb command should be executed frequently to ensure the consistency of the TMR's database. However, you should always use this command before performing a wbkupdb. Unfortunately, this command can run quite slowly, depending on the number of Managed Nodes. Again, LCF will help reduce the elapsed time required to perform a wchkdb command.

A couple of tips for using wchkdb in a smart way:

1. wchkdb -o <output to a file>

This lists the discrepancies found into the output file without attempting to fix them.

2. wchkdb -u -f <same file as before>

This will pass the list of discrepancies to the master daemon that will call only the involved methods for correcting them.

You might also investigate using the wchknode command. This will check a subset of the database for a particular node and will run quicker than wchkdb.

---

## 9.2 Removing Managed Nodes

To remove a Managed Node from your TMR, follow these steps:

1. Login to the TMR Server as root.
2. Create a backup of the object database.
3. Delete the Managed Node from the name registry by using the GUI, or the wrmnode command.

Later in this chapter, we also provide a script mn\_rm.ksh that automates this process. It is described in 9.5.2, "Deleting a Managed Node" on page 131.

4. Run wchkdb -u.

**For Windows NT only:**

5. Login to the local machine and setup the Tivoli environment.  
`\winnt\system32\drivers\etc\Tivoli\setup_env.cmd`
6. Remove the oserv from the service manager. The oinstall -remove command removes the Tivoli object dispatcher entry from the Services panel. It also removes the registry entry HKEY\_LOCAL\_MACHINE\SOFTWARE\Tivoli\Platform\oserv-xx where xx is the port number (usually 94)



7. Optionally, you can also remove TRIP. The `trip -remove` removes TRIP from the Services panel and the registry entry

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Trip
```

**For UNIX only:**

8. Login to the local machine and setup the Tivoli environment.  

```
. /etc/Tivoli/setup_env.sh
```
9. Verify if the `oserv` is stopped.  

```
# ps -ef | grep oserv
```
10. If the `oserv` is not yet stopped, run  

```
. /etc/Tivoli/oserv.rc stop
```
11. Delete the Tivoli database and binaries manually. By default, the object database is in `\var\spool\Tivoli\xxx.db` where `xxx` is the hostname of the machine. The binaries are in `\usr\local\Tivoli`. The subdirectories are `bin`, `lib`, `man`, and `msg_cat`.
12. Edit the files `/etc/rc.nfs`, `/etc/services` and `inetd.conf` to remove references to Tivoli.

---

## 9.3 Transfer TME 10 CD Images to Disk

If you will be installing multiple TME 10 products on multiple systems, you may want to copy the TME 10 Product CDs to a file server. Then any system that requires access to the installation files can simply mount or otherwise access the shared disk.

TME 10 provides a command called `wpcdrom` to copy the CD-ROM image. However, if the `-c` parameter is not used, the files are not copied. Rather, symbolic links are created. If a copy is desired, ensure you use the `-c` parameter.

On some platforms (such as AIX), the `wpcdrom` command copies the files to the target disk with lower case names. This is due to the CD-ROM device drivers that are used. However, the installation process expects files to be in upper case, so you will need to rename the files on your newly copied image to their upper case counterparts.

### 9.3.1 Using `file0.tar`

The file `file0.tar` is created when you use the procedure in section 9.3, “Transfer TME 10 CD Images to Disk” when creating an install image of the TME 10 Framework on your hard disk. This file takes the place of running `WPREINST.SH` every time you need to re-install the Server. Simply `untar` the file in a temporary install directory and run the `wserver` command.

```
# mkdir /usr/local/Tivoli/install
# cd /usr/local/Tivoli/install
# cp /inst.images/TME_3.1.C/file0.tar .
# tar -xvf file0.tar
```

Run the wserver command to re-install the TMR Server.

```
# ./wserver -c /inst.images/TME_3.1.C
```

---

## 9.4 Troubleshooting Client Installation

Below are some of the more common problems you may run into when installing a Managed Node, as well as the solutions to them.

1. If you are using a non-English keyboard, you may run into the following error messages:

```
$database mismatch ( from triton.tivoli.com/146.84.26.6)
$migrating ALI
$changing encryption type from none to simple
TME 10 Framework () date time copyright etc
TMR 1724398290 ORB 1 TMR server local:94 port 94
Hmm.. looks like you're running NT 4.0 Service pck 1
I won't create a console
creating seperate windowstation and desktop
creatingwindowstation failed:the specified module not found
Hmm....(these messages then repeat indefinitely)
.....
```

This is a known problem with the CreateWindowStation() call. To fix this:

- a. Open the Control Panel.
- b. Select the keyboard icon.
- c. Select Input Locales.
- d. Add English(United States).
- e. You do not need to set this as the default locale; it just needs to be in the list.

Note: You need to re-boot the machine to have the change take effect.

2. When specifying the user ID for an NT system to be installed as a Managed Node, ensure you specify Administrator with a capital A. Otherwise, the installation will not succeed.
3. Tivoli setup requests a path name. For NT targets, this cannot include long names. Setup will fail immediately without warning.

---

## 9.5 Miscellaneous Scripts

The rest of this chapter consists of a series of scripts that we have found helpful in our environment. These are not officially supported and you should thoroughly understand their content before using them in your own environment.

### 9.5.1 Finding out Who's Running the TME 10 Desktop

The script below displays the current Administrators using Tivoli. The name of the script is `whoson`.

```
#!/bin/sh -e
# This is unsupported software.
# Author: Steve Cochran (stevec@tivoli.com)

ADL=wlookup -r ActiveDesktopList adl

# Get the list of current entries in the ActiveDesktopList
ENTRIES=idlcall $ADL _get_entries
N_ENTRIES=idlarg 1 $ENTRIES

echo "Host                Administrator"
echo "-----"

# the 'first' entry is the number of entries in the list
# increment our counters by 1
i=2;
N_ENTRIES=expr $N_ENTRIES + 1

# Go through the list and print out the host and the label of the administrator
# associated with each entry.
while [ expr $i -le $N_ENTRIES ]; do
    THIS=idlarg $i $ENTRIES

    HOST=idlarg 1 $THIS | awk -F\" '{print $2}'
    ADMIN=idlcall \idlarg 3 $THIS\ _get_label | awk -F\" '{print $2}'

    echo "$HOST          $ADMIN"

    i=expr $i + 1
done

exit 0
```

### 9.5.2 Deleting a Managed Node

This script, called `mn_rm.ksh`, allows you to cleanly remove a Managed Node from the TME name registry and object database.

```
#!/bin/ksh
#
. /etc/Tivoli/setup_env.sh

IO=wlookup Installation
read MN? Enter Managed Node Name: '
echo "Deletion of $MN started on date" >> /tmp/ManagedNode_Deletes
wrmnode -f $MN
```

```

idlcall $IO remove_host_locations '{1 ""$MN}'
AppList=wlookup -ar ProductInfo | awk '{print $2}'
for i in $AppList
do
    idlcall $i remove_host_locations '{1 ""$MN}'
done
PatchList=wlookup -ar PatchInfo | awk '{print $2}'
for i in $PatchList
do
    idlcall $i remove_host_locations '{1 ""$MN}'
done
print "Removed Managed Node: $MN"
echo "$MN deleted on date" >> /tmp/ManagedNode_Deletes

```

### 9.5.3 Backup the TMR

This script, called Backup\_TMR.ksh, allows you to create a backup of the available Managed Nodes in your TMR. It will skip the backup of unavailable Managed Nodes.

This script can be very useful in scheduling your TMR backup, allowing you to continue with the backup even if some of the Managed Nodes are unavailable.

```

#!/bin/ksh
#

# wlookup for all managed nodes, then wping them, and do a wbackup, but
# only on the node which answered on the wping.
#
# If you give it an argument, it will prepend it to the name of the backup file
#
# Author: Yves Dorfsman, Jan 14th 1997
#

. /etc/Tivoli/setup_env.sh

OPERATION=$1
[[ -n "$OPERATION" ]] && OPERATION="${OPERATION}_"
ALL_NODES=""
DATESTAMP=$(date "+DB_ %b%d-%H%M")
BACKUPDIR=$DBDIR/./backups

# Find out all the managed nodes, and wping them
for i in $(wlookup -ar ManagedNode |cut -f1)
do
    wping $i
    if (($? == 0)) ; then
        ALL_NODES="${ALL_NODES} $i"
    fi
done

wbkupdb -d ${BACKUPDIR}/${OPERATION}${DATESTAMP} $ALL_NODES

exit $?

```

## 9.5.4 Installing a Full TMR

The next several scripts were used in our test environment to install a full TMR piece by piece. We first install the TMR Server, then Managed Nodes, followed by PC Managed Nodes and finally a set of TME 10 applications. These scripts are relatively lengthy, but you may find useful techniques among them for installing TMRs in complex environments.

### 9.5.4.1 Installing the TME 10 Framework

If you are going to be installing TMR Servers on a variety of systems (such as in a lab environment), where you may need to specify many options, a script such as the following may be of use to you.

```
#!/bin/ksh
# Name: inst_tmr
# Creates the TMR Server
# Author: Armand Adriano, IBM
#
# Modified by: Fabrizi Francesco      @FF1 6/6/97

# @FF1 add Start
clear
. tmr_functions
{
  while read lin; do
    lin=${lin## }
    lin=${lin%% }
    if [[ -n $lin ]]; then
      Key=echo $lin|cut -f1 -d:
      Value=echo $lin|cut -f2 -d:
      Value=${Value## }
      case $Key in
        BASEDIR )
          MBASE=$Value;;
        DBDIR )
          MALIDB=$Value;;
        SOURCEDIR )
          MSOURCE_DIR=$Value;;
        LOGDIR )
          MLOGDIR=$Value;;
        KEY )
          MLK=$Value;;
        * )
          print ">>> Keyword $Key skipped.";;
      esac
    else
      echo "Skip "
    fi
  done

} < tme.cfg

MBASE=${MBASE:? "missing value!"}
MALIDB=${MALIDB:? "missing value!"}
MSOURCE_DIR=${MSOURCE_DIR:? "missing value!"}
MLK=${MLK:? "missing value!"}
MLOGDIR=${MLOGDIR:? "missing value!"}
```

```

MLOGFILE="$MLOGDIR/$0.log"

echo "***** Log File *****">$MLOGFILE
echo "Configuration Parameters:">>$MLOGFILE

T_INPUT=0                # Initial status
T_COMPLETE=0            # syntax is complete 1 it does not complete 0
T_COUNTER=0              # Variable counter
T_QUESTION=""
if [[ $# = 0 ]] then
    T_INPUT=2            # Intercative mode
else
    while getopts ":r:yh" opt; do
        case $opt in
            r )
                T_INPUT=1
                Regname=$OPTARG
                let T_COUNTER=T_COUNTER+1;;
            y )
                T_INPUT=1
                T_QUESTION="y";;
            \? )
                print "\n*** ERROR *** Unknown option see inst_tmr -h for more \
help" ;;
            h )
                print "\nTME10 Managed Node installation"
                print "  The syntax is"
                print " "
                print "  inst_tmr -r region [-y]"
                print "  inst_mmode "
                return 0;;
        esac
    done
fi

if [[ $T_INPUT = 1 && $T_COUNTER = 1 ]] then
    T_OK=1
elif [[ $T_INPUT = 2 ]] then
    T_OK=1
else
    print "\n*** ERROR *** Syntax Error see inst_tmr -h for more helps"
    return 1
fi

if [[ $T_INPUT = 2 ]] then
    echo `TMR Installation`
    echo `nRegion Name: \c`; read Regname
    echo "Do you want install/reinstall the TMR Server really? \c"
    read T_QUESTION
fi

ExistDir $MSOURCE_DIR
if [[ $? = 1 ]] then
    return $?
fi

if [[ $T_QUESTION != "Y" && $T_QUESTION != "y" ]]; then
    echo "Do you want erase the TMR Server really? \c"
    read T_QUESTION
fi
# @FF add Stop

```

```

#-----START MODIFICATION HERE-----
# set up hard-coded variables
MIP=""
INDF="TMF"

# set up floating variables
MHOST=uname -n
DATESTAMP=$(date "+%m%d%H%M")

#set up fixed variables
# Temporary directory
MINSTALL_DIR="$MBASE/install"
# Binary Code Directory
MBIN="$MBASE/bin"
# Runtime library Directory
MLIB="$MBASE/lib"
# Manuals and ...
MMAN="$MBASE/man"
# Catalog directories
MCAT="$MBASE/msg_cat"
# ... and logs file directory
MCINSTALL="$MALIDB/cinstall"

# Region Name (Input Field)
MRN=$Regname
# Must the Framework will be started automatically? 1(Yes) | 0(No)
MAutoStart=1
# Must the Tivoli Daemon will be started remotely? 1(Yes) | 0(No)
MSetPort=1
# Must the product directories will be created automatically? 1(Yes) | 0(No)
MCreatePaths=1

#-----END MODIFICATION HERE-----

export DOGUI=1

rem_tmr -$QUESTION

# If the installation directory does not exist create it.
if [[ ! -d $MINSTALL_DIR ]]
then
    mkdir $MINSTALL_DIR
fi

cd $MINSTALL_DIR

# Uncomment the tar command below if you used tar -chf .... after wpcdrom
# tar xf $MSOURCE_DIR/file0.tar
# Otherwise, uncomment the WPREINST.SH command below if you used a script
# to convert the files from lower case to upper case
$MSOURCE_DIR/WPREINST.SH

./wserver -y -c $MSOURCE_DIR BIN=$MBIN! LIB=$MLIB! ALIDB=$MALIDB! MAN=$MMAN! \
CAT=$MCAT! RN=$MRN IP=$MIP LK=$MLK AutoStart=$MAutoStart SetPort=$MSetPort \
CreatePaths=$MCreatePaths

# Create if the Backup log directory does not exist
if [[ ! -d $MCINSTALL ]]
then

```

```

    mkdir $MCINSTALL
fi
# Copy the logs file into the backup directory
cp /tmp/tivoli.sinstall $MCINSTALL/$MHOST'_sintstall_'$INDF'.'$DATESTAMP
return 0

```

### 9.5.4.2 Installing Managed Nodes

Similar to the previous script, this script will install multiple Managed Nodes in a complex environment.

```

#!/bin/ksh
# Name: inst_mnode
# Creates and install TMR Managed Nodes
# Authors:
#         ITSO LCF Team IBM Austin Tx
# This scripts allows you to install one or more Managed Nodes
# you can use it in several way.
# You can use it interactively submitting the command inst_mnode or
# use it throughout an input file containing all the client that must be
# installed. The format of the files is:
#
# First row contains this kind of information
#   <Administrator>:<TMR Name>:<Password>
# Next lines contain
#   <Node Name>
#
# Warnings:
#   You cannot put blank line between the data.
#   You must put each client for row.
#   All the client must have defined the same Administrator
#   and the same password.
#
#   To use this scripts you must define first of all a file called
#   tme.cfg containing some basic information.
#   The possible keys in this file are:
#       BASEDIR: <Tivoli Binary Dir location>
#       DBDIR: <DB Location>
#       SOURCEDIR: <Sorce file location>
#       LOGDIR: <Where put the log file>
#       KEY: <Key Number>
#
#   All the key are mandatory except the key KEY that is used only to
#   install the TMR Server.

clear
. tmr_functions
{
    while read lin; do
        lin=${lin## }
        lin=${lin%% }
        if [[ -n $lin ]]; then
            Key=echo $lin|cut -f1 -d:
            Value=echo $lin|cut -f2 -d:
            Value=${Value## }
            case $Key in
                BASEDIR )
                    MBASE=$Value;;
                DBDIR )

```



```

        MALIDB=$Value;;
        SOURCEDIR )
        MSOURCE_DIR=$Value;;
        LOGDIR )
        MLOGDIR=$Value;;
        * )
#         print ">>> Keyword $Key skipped.>";;
    esac
    else
        echo "Skip "
    fi
done

} < tme.cfg

MBase=${MBase:? "missing value!"}
MALIDB=${MALIDB:? "missing value!"}
MSOURCE_DIR=${MSOURCE_DIR:? "missing value!"}
MLOGDIR=${MLOGDIR:? "missing value!"}

MLOGFILE="$MLOGDIR/$0.log"

echo "***** Log File *****">$MLOGFILE
echo "Configuration Parameters:">>$MLOGFILE

T_INPUT=0           # Initial status
T_COMPLETE=0       # syntax is complete 1 it does not complete 0
T_COUNTER=0        # Variable counter
if [[ $# = 0 ]] then
    T_INPUT=3       # Interactive mode
else
    while getopts ":r:n:f:hu:" opt; do
        case $opt in
            f )
                if [[ $T_INPUT = 0 ]] then
                    T_INPUT=1 # Input from a file
                    FileName=$OPTARG
                else
                    print "\n*** ERROR *** Input mismatch see inst_mmode -h for \
more help"
                    return 1
                fi;;
            n )
                T_INPUT=2 # Input from Command line
                ClientName=$OPTARG
                let T_COUNTER=T_COUNTER+1;;
            r )
                T_INPUT=2
                Regname=$OPTARG
                let T_COUNTER=T_COUNTER+1;;
            u )
                T_INPUT=2
                User=$OPTARG
                let T_COUNTER=T_COUNTER+1;;
            \? )
                print "\n*** ERROR *** Unknown option see inst_mmode -h for more \
help" ;;
            h )
                print "\nTME10 Managed Node installation"
                print "  The syntax is"
                print " "

```

```

                print "    inst_mmode -f input_file - Input from a list"
                print "    inst_mmode -n name -r region -u user \
- Input from Command Line"
                print "    inst_mmode - Interactive"
                return 0;;
            esac
        done
    fi

    if [[ $T_INPUT = 1 && $T_COUNTER = 0 ]] then
        T_OK=1
    elif [[ $T_INPUT = 2 && $T_COUNTER = 3 ]] then
        T_OK=1
    elif [[ $T_INPUT = 3 ]] then
        T_OK=1
    else
        print "\n*** ERROR *** Syntax Error see inst_mmode -h for more helps"
        return 1
    fi

    if [[ $T_INPUT = 3 ]] then
        echo 'TMR Managed Node Installation'
        echo '\nRegion Name: \c'; read Regname
        echo 'Managed Node Name: \c'; read ClientName
        echo '\nInput User: \c'; read User
    fi

    ExistDir $MSOURCE_DIR
    if [[ $? = 1 ]] then
        return $?
    fi

    # Temporary directory
    MINSTALL_DIR="$MBASE/install"
    # Binary Code Directory
    MBIN="$MBASE/bin"
    # Runtime library Directory
    MLIB="$MBASE/lib"
    # Manuals and ...
    MMAN="$MBASE/man"
    # Catalog directories
    MCAT="$MBASE/msg_cat"
    # logs file directory
    MCINSTALL="$MALIDB/cinstall"

    echo " Base Directory.....$MBASE" >> $MLOGFILE
    echo "   Installation Dir.....$MINSTALL_DIR" >> $MLOGFILE
    echo "   Binary Dir.....$MBIN" >> $MLOGFILE
    echo "   Library Dir .....$MLIB" >> $MLOGFILE
    echo "   Documentation Dir.....$MMAN" >> $MLOGFILE
    echo "   Catalog Dir .....$MCAT" >> $MLOGFILE
    echo " Database Directory.....$MALIDB" >> $MLOGFILE
    echo "   Tivoli Log Dir.....$MCINSTALL" >> $MLOGFILE
    echo "   Source Directory.....$MSOURCE_DIR" >> $MLOGFILE
    echo "   Log Directory .....$MSOURCE_DIR" >> $MLOGFILE
    echo "   log File.....$MLOGFILE" >> $MLOGFILE
    echo " " >> $MLOGFILE
    echo "The parameters are:" >> $MLOGFILE

    if [[ $T_INPUT = 1 ]] then
        COUNTER=0

```

```

ListClient=""
{
  while read lin; do
    if [[ $COUNTER = 0 ]] then
      User=echo $lin|cut -f1 -d:
      User=${User## }
      User=${User%% }
      Regname=echo $lin|cut -f2 -d:
      Regname=${Regname## }
      Regname=${Regname%% }
      let COUNTER=COUNTER+1
      echo " User.....$User" >> $MLOGFILE
      echo " Region.....$Regname" >> $MLOGFILE
      echo " Clients to install:" >> $MLOGFILE
    else
      tClient=echo $lin|cut -f1 -d:
      tClient=${tClient## }
      tClient=${tClient%% }
      echo " $tClient" >> $MLOGFILE
      ListClient="$ListClient $tClient"
    fi
  done
} < $FileName
ListClient=${ListClient## }
echo "wclient -d -p $Regname -U $User -y -c $MSOURCE_DIR BIN=$MBIN! LIB=$MLIB! ALIDB=$MALIDB! \
MAN=$MMAN! CAT=$MCAT! AutoStart=$MAutoStart SetPort=$MSetPort CreatePaths=$MCreatePaths \
$listClient" >> $MLOGFILE
echo "Please Wait. The installation is running..."
wclient -d -p $Regname -U $User -y -c $MSOURCE_DIR BIN=$MBIN! LIB=$MLIB! ALIDB=$MALIDB! \
MAN=$MMAN! CAT=$MCAT! AutoStart=$MAutoStart SetPort=$MSetPort \
CreatePaths=$MCreatePaths $ListClient
rc=$?
else
echo " User.....$User" >> $MLOGFILE
echo " Region.....$Regname" >> $MLOGFILE
echo " Client to install: $ClientName" >> $MLOGFILE
echo "wclient -d -p $Regname -U $User -y -c $MSOURCE_DIR BIN=$MBIN! LIB=$MLIB! ALIDB=$MALIDB! \
MAN=$MMAN! CAT=$MCAT! AutoStart=$MAutoStart SetPort=$MSetPort CreatePaths=$MCreatePaths \
$ClientName" >> $MLOGFILE
echo "Please Wait. The installation is running..."
wclient -d -p $Regname -U $User -y -c $MSOURCE_DIR BIN=$MBIN! LIB=$MLIB! ALIDB=$MALIDB! \
MAN=$MMAN! CAT=$MCAT! AutoStart=$MAutoStart SetPort=$MSetPort \
CreatePaths=$MCreatePaths $ClientName
rc=$?
fi

if [[ $rc = 0 ]] then
  echo "Installation ended!"
  echo "Installation ended!" >> $MLOGFILE
else
  echo "Client Installation has returned rc = $rc"
  echo "Client Installation has returned rc = $rc" >> $MLOGFILE
fi
echo "***** Log File *****">>$MLOGFILE
return $rc

```

### 9.5.4.3 Installing PC Managed Nodes

Similar to the previous scripts, this script installs PC Managed Nodes.

```

#!/bin/ksh
# Name: inst_pcmnode
# Creates a TMR PC Managed Node
# Authors:
#         ITSO LCF Team IBM Austin Tx

# This scripts allows you to register one or more PcManaged Nodes
# you can use it in several way.
# You can use it interactively submitting the command inst_pcmnode or
# use it throughout an input file containing all the client that must be
# installed. The format of the files is:
#
# First row contains this kind of information
#   <Administrator>:<TMR Name>:<Password>
#   Only the second of the first three information is needed to register
#   the node the other two are kept for compatibility with the structure
#   of the file used to install the Manged Nodes.
# Next lines contain
#   <Node Name>:<Node Type>
#
# Warnings:
#   You cannot put blank line between the data.
#   You must put each client for row.
#   The <Node Type> could not be present. It means that the Node must
#   be on line. The possible values allowed for the <Node Type> are:
#       - dos
#       - netware
#       - nt
#       - windows
#
#   To use this scripts you must define first of all a file called
#   tme.cfg containing some basic information.
#   The possible keys in this file are:
#       BASEDIR: <Tivoli Binary Dir location>
#       DBDIR:   <DB Location>
#       SOURCEDIR: <Sorces file location>
#       LOGDIR:  <Where put the log file>
#       KEY:    <Key Number>
#
#   In the PcManaged Node installation only the LOGDIR key is
#   mandatory. The other are mandatory to install the Framework and
#   to install the Managed Node.
#

clear
. tmr_functions
{
  while read lin; do
    lin=${lin## }
    lin=${lin%% }
    if [[ -n $lin ]]; then
      Key=echo $lin|cut -f1 -d:
      Value=echo $lin|cut -f2 -d:
      Value=${Value## }
      case $Key in
        LOGDIR )
          MLOGDIR=$Value;;
        * )
          print ">>> Keyword $Key skipped.";;
      esac
    else

```

```

        echo "Skip "
    fi
done

} < tme.cfg

MLOGDIR=${MLOGDIR:? "missing value!"}

MLOGFILE="$MLOGDIR/$0.log"

echo "***** Log File *****">$MLOGFILE
echo "Configuration Parameters:">>$MLOGFILE

T_INPUT=0           # Initial status
T_COMPLETE=0       # syntax is complete 1 it does not complete 0
T_COUNTER=0        # Variable counter
if [[ $# = 0 ]] then
    T_INPUT=3       # Intercative mode
else
    while getopts ":r:n:f:ht:" opt; do
        case $opt in
            f )
                if [[ $T_INPUT = 0 ]] then
                    T_INPUT=1 # Input from a file
                    FileName=$OPTARG
                else
                    print "\n*** ERROR *** Input mismatch see inst_mmode -h for \
more help"
                    return 1
                fi;;
            n )
                T_INPUT=2 # Input from Command line
                ClientName=$OPTARG
                let T_COUNTER=T_COUNTER+1;;
            t )
                T_INPUT=2
                Type=$OPTARG
                let T_COUNTER=T_COUNTER+1;;
            r )
                T_INPUT=2
                Regname=$OPTARG
                let T_COUNTER=T_COUNTER+1;;
            \? )
                print "\n*** ERROR *** Unknown option >$opt< see inst_mmode -h for \
more help" ;;
            h )
                print "\nTME10 Managed Node installation"
                print "  The syntax is"
                print " "
                print "    inst_mmode -f input_file  - Input from a list"
                print "    inst_mmode -n name -t type -r region \
- Input from Command Line"
                print "    inst_mmode                - Interactive"
                print " "
                print "      where type could be:"
                print "          dos"
                print "          netware"
                print "          nt"
                print "          windows"
                return 0;;
        esac
    done

```

```

done
fi

if [[ $T_INPUT = 1 && $T_COUNTER = 0 ]] then
    T_OK=1
elif [[ $T_INPUT = 2 && $T_COUNTER = 3 ]] then
    T_OK=1
elif [[ $T_INPUT = 3 ]] then
    T_OK=1
else
    print "\n*** ERROR *** Syntax Error see inst_pcmnode -h for more helps"
    return 1
fi

if [[ $T_INPUT = 3 ]] then
    echo 'TMR Managed Node Installation'
    echo '\nRegion Name: \c'; read Regname
    echo 'PcManagedNode Name: \c'; read ClientName
    echo 'PcManagedNode Type: \c'; read Type
fi

if [[ $T_INPUT = 1 ]] then
    COUNTER=0
    {
        while read lin; do
            if [[ $COUNTER = 0 ]] then
                User=echo $lin|cut -f1 -d:
                User=${User## }
                User=${User%% }
                Regname=echo $lin|cut -f2 -d:
                Regname=${Regname## }
                Regname=${Regname%% }
                let COUNTER=COUNTER+1
                echo " Region.....$Regname" >> $MLOGFILE
                echo " Clients to install:" >> $MLOGFILE
                echo " Registering PcManaged Nodes..."
            else
                tClient=echo $lin|cut -f1 -d:
                tType=echo $lin|cut -f2 -d:
                tClient=${tClient## }
                tType=${tType## }
                tClient=${tClient%% }
                tType=${tType%% }
                if [[ $tType = $tClient ]]; then
                    tType=""
                fi
                echo " $tClient/$tType" >> $MLOGFILE
                echo " $tClient/$tType ...c"
                if [[ -n $tType ]]; then
                    temp="-p $tType"
                else
                    temp=""
                fi
                wcrtpcmngnode $temp $tClient $Regname >> $MLOGFILE 2>&1
                rc=$?
                if [[ $rc = 0 ]] then
                    echo " registered"
                else
                    echo " *** ERROR rc = $rc"
                    echo " *** ERROR rc = $rc" >> $MLOGFILE
                fi
            fi
        done
    }
fi

```

```

        fi
    done
} < $FileName
else
echo " Region.....$Regname" >> $MLOGFILE
echo " Client to install/Type: $ClientName/$Type" >> $MLOGFILE
echo " Registering PcManaged Node \c"
echo "$ClientName/$Type...\c"
if [[ -n $Type ]]; then
    temp="-p $Type"
else
    temp=" "
fi
wcrtpcmngnode $temp $ClientName $Regname >> $MLOGFILE 2>&1
rc=$?
if [[ $rc = 0 ]] then
    echo " registered"
else
    echo " *** ERROR rc = $rc"
    echo " *** ERROR rc = $rc" >> $MLOGFILE
fi
fi
echo "Change ended!"
echo "Change ended!" >> $MLOGFILE
echo "***** Log File *****">>$MLOGFILE
return $rc

```

#### 9.5.4.4 Personalizing the Environment

We used the following script to personalize the environment after having installed the TMR.

```

#!/bin/ksh!
# Name: BuildEnv
# Test Environment Definition
# IBM Corporation
# authors:
#     Fabrizi Francesco - IBM Italy
#     Rossi Renata     - IBM Italy
# this script create the Test Environment for our LCF Laboratory,
# add Policy Regions and Profile Managers
#
clear

# Variables Definition
BaseTMR=Bedrock
ChildTMR1=COFFEE
ChildTMR2=LABs
ChildTMR3=TEA
ProfileMg1=NTGROUP
ProfileMg2=AIXPLATFORM
ProfileMg3=SENTRY
ProfileMg4=COURIER
ProfileMg5=SENTRYAIX

if [[ -z $1 ]]; then
    echo "TME 10 Framework ver. 3.2"
    echo "Test Environment Definition"
    echo "Syntax:"
    echo "\n BuildEnv <Administrator>"

```

```

echo "\n  where:"
echo "    <Administrator> is a specific Administrator's Desktop"
echo "\n"
else
# I create the main TMR ...
echo Creating Base TMR in the desktop $1...
wrtpr -a $1 $BaseTMR
echo ... creating into $BaseTMR the first Sub-Tmr ...
wrtpr -s @$BaseTMR $ChildTMR1
echo ... creating the second Sub-TMR and ...
wrtpr -s @$BaseTMR $ChildTMR2
echo ... creating the third Sub-TMR.
wrtpr -s @$BaseTMR $ChildTMR3
echo Adding the resource ProfileManager into TMRs
wsetpr ProfileManager @PolicyRegion:BaseTMR
wsetpr ProfileManager @PolicyRegion:ChildTMR1
wsetpr ProfileManager @PolicyRegion:ChildTMR2
wsetpr ProfileManager @PolicyRegion:ChildTMR3
echo Creating the ProfileManagers...
wrtprfmgr @$BaseTMR $ProfileMg1
wrtprfmgr @$BaseTMR $ProfileMg2
wrtprfmgr @$BaseTMR $ProfileMg3
wrtprfmgr @$ChildTMR3 $ProfileMg4
wrtprfmgr @$ChildTMR2 $ProfileMg5
echo Refreshing the Desktop
wrefresh /Administrators/$1
fi

```

#### 9.5.4.5 Installing Applications

After installing the TMR and customizing the environment, the next step is to install a set of applications. This script automates this phase of installing our environment.

```

#!/bin/ksh
# Name: inst_prod
# Installs products and patches on ManagedNodes.

# Armand Adriano, IBM

#This script reads from standard input names of a file containing
#the list of product to install ( prod_list_3.1.txt) and a filename of
#patches to install ( patch_list_3.1.txt ) and the nodes where to install on
#
#HOW TO USE THIS SCRIPT
#
#you have to modify the files *.txt in order to install any product you
#need: <prod_description>|<prod_directory>|<prod_name>
#i.e. TME 10 Software Distribution Release 3.1|SoftDist.3.1.RevA|COURIER
#
#<prod_description> is in documented in the *.IND file 1st line
#<prod_name> is the name of the *.IND file that you see in the product
#directory, each product may have more than one .IND file
#
#
#adding the INSTDIR variable for getting different products
#from different machines
#INSTDIR is the mount point of your cdrom or the directory
#where you copied the product images

```



```

#
#You can change the INSTDIR variable
#INSTDIR=tivprods
INSTDIR=mnt

list_prod()
{
echo '\nProducts:\n '
c=1
while [ c -le $NUMPROD ]
do
    echo ${PRODNAME[c]}
    c=expr $c + 1
done
}

list_patch()
{
echo '\nPatches:\n '
c=1
while [ c -le $NUMPATCH ]
do
    echo ${PATCH[c]}
    c=expr $c + 1
done
}

# Main Body

. /etc/Tivoli/setup_env.sh
cd $DBDIR/..;RT=pwd
MCINSTALL="$RT/cinstall"
mcinstall="/tmp/tivoli.cinstall"
DATESTAMP=$(date "+%m%d%H%M")
SOURCE="/home/scripts/RE"

clear
echo 'Products and patches installation.\n\n'
echo "Enter Product List File at $SOURCE :c"; read PROD_LIST
echo "Enter Patch List File at $SOURCE :c"; read PATCH_LIST
echo 'Enter target nodes (space-separated): \c'
read INPUT
NODE=""
for i in $INPUT
do
    wping $i
    if [ $? -eq 0 ]
    then NODE="$NODE $i "
    echo "$i reached."
    else
    echo "$i not responding ..."
    fi
done

if (($? != 0)); then
    echo "Node $i not responding."
    echo "Aborting...."
    return 0
fi

```

```

echo "\n\nThe following will be installed on $NODE"
c=1
while read x
do
    PRODFNAME[c]=$(print "$x" | cut -f1 -d\|)
    DIR[c]=$(print "$x" | cut -f2 -d\|)
    INDF[c]=$(print "$x" | cut -f3 -d\|)
    c=expr $c + 1
done < $SOURCE/$PROD_LIST

NUMPROD=expr $c - 1
list_prod

c=1
while read x
do
    PATCH[c]=$(print "$x" | cut -f1 -d\|)
    PATCHDIR[c]=$(print "$x" | cut -f2 -d\|)
    PATCHINDF[c]=$(print "$x" | cut -f3 -d\|)
    c=expr $c + 1
done < $SOURCE/$PATCH_LIST

NUMPATCH=expr $c - 1
list_patch

echo '\n\nPress Y to continue, any other key to abort. \c'; read RESP
if [ $RESP != 'Y' ]; then
    echo "\n\nBye..."
    return 0
fi
echo "\n\nHere we go ..."
c=1
while [ c -le $NUMPROD ]
do
    # rm $mcininstall
    # winstall -y -c /$INSTDIR/${DIR[c]} -i ${INDF[c]} $NODE
    # sleep 2
    # cp $mcininstall $MCMINSTALL/$NODE'_cinstall'_${INDF[c]}'. '$DATESTAMP
    c=expr $c + 1
done

c=1
while [ c -le $NUMPATCH ]
do
    # wpatch -y -c /$INSTDIR/${PATCHDIR[c]} -i ${PATCHINDF[c]} $NODE
    # mv $mcininstall $MCMINSTALL/$NODE'_cinstall'_${PATCHINDF[c]}'. '$DATESTAMP
    c=expr $c + 1
done

echo 'Done.'
return 0

```

#### 9.5.4.6 Populate the Profile Managers

This script was used to populate our profile managers based on operating system.

```

#!/bin/ksh!
# Name: pm_pop
#
# 1997/07/07 IBM Corporation
# authors:
#         IBM ITSO Residents
#
# Description:
#         Populates ProfileManagers with the ManagedNodes and
#         PcManagedNodes filtered the operating system
#
# Parameters:
#         <ResourceType> possible value ManagedNode or PcManagedNode"
#
clear

# Variables Definition
MaxOS=5                # lenght of the operating system Array
#
# The array OS links the operating system with a specific ProfileManager.
# Different string could be represent the same Op.Sys. in the following
# array "Windows_NT" and "Windows NT" means the some thing. The reason
# for that is due to the fact that the output for the details about
# the PcManagedNode and the ManagedNode are completely different
# and the Op.Sys. representation is coded differently.
OS[0]="AIX:AIXPLATFORM"
OS[1]="Windows_NT:NTGROUP"
OS[2]="Windows NT:NTGROUP"
OS[3]="Windows 95:95GROUP"
OS[4]="OS/2:OS2GROUP"

OSTot=""
let x=0
while (( x < $MaxOS - 2 ))
do
    OSTot="$OSTot${OS[$x]},"
    let x=x+1
done
OSTot="$OSTot${OS[($MaxOS - 1)]}"

if [[ -z $1 ]]; then
    echo "TME 10 Framework ver. 3.2"
    echo "Populates the ProfileManagers"
    echo "Syntax:"
    echo "    pm_pop <Resource_type>"
    echo "    where:"
    echo "        <Resource_Type> could be:"
    echo "                ManagedNode"
    echo "                PcManagedNode"
    echo "\n"
else
    echo "Adding $1..."
    wgetallinst $1 > t 2>&1                # ask for all the installed objects
                                           # that their type is $1

    if [[ $? = 0 ]]; then
    {
        while read lin; do                # for each object ask for the details
            echo "    $lin...\c"
            if [[ $1 = "ManagedNode" ]]; then
                wmannode $lin > t1 2>&1    # details for the ManagedNode
            elif [[ $1 = "PcManagedNode" ]]; then
            }
    }

```

```

        wpcmgnode $lin > t1 2>&1 # details for the PcManagedNode
    else
        echo "**** ERROR - the resource type $1 cannot be managed"
        exit
    fi
    if [[ $? = 0 ]]; then
        awk -f add2pfm.awk "resource=$1" "node=$lin" "subs=$OSTot" t1
    else
        echo "**** ERROR - $1 $lin does not exist"
        exit
    fi
    rm t1
    echo "added"
done
} < t
#refresh
else
    echo "**** ERROR - Resource Type Unknown"
    exit
fi
fi
rm t

```

Actually, the populate profile manager procedure used an AWK script, which is shown here:

```

#!/bin/ksh!
# NameL add2fpm.awk
#
# 1997/07/07 IBM Corporation
# authors:
#     IBM ITSO Residents
#
# Description:
#     Populates ProfileManagers with the ManagedNodes and
#     PcManagedNodes filtered the operating system
#
# Parameters:
#     <ResourceType> possible value ManagedNode or PcManagedNode"
#
clear

# Variables Definition
MaxOS=5 # length of the operating system Array
#
# The array OS links the operating system with a specific ProfileManager.
# Different string could be represent the same Op.Sys. in the following
# array "Windows_NT" and "Windows NT" means the some thing. The reason
# for that is due to the fact that the output for the details about
# the PcManagedNode and the ManagedNode are completely different
# and the Op.Sys. representation is coded differently.
OS[0]="AIX:AIXPLATFORM"
OS[1]="Windows_NT:NTGROUP"
OS[2]="Windows NT:NTGROUP"
OS[3]="Windows 95:95GROUP"
OS[4]="OS/2:OS2GROUP"

OSTot=""
let x=0

```

```

while (( x < $MaxOS - 2 ))
do
    OSTot="$OSTot${OS[$x]},"
    let x=x+1
done
OSTot="$OSTot${OS[($MaxOS - 1)]}"

if [[ -z $1 ]]; then
    echo "TME 10 Framework ver. 3.2"
    echo "Populates the ProfileManagers"
    echo "Syntax:"
    echo "  pm_pop <Resource_type>"
    echo "  where:"
    echo "      <Resource_Type> could be:"
    echo "          ManagedNode"
    echo "          PcManagedNode"
    echo "\n"
else
    echo "Adding $1..."
    wgetallinst $1 > t 2>&1          # ask for all the installed objects
                                    # that their type is $1

    if [[ $? = 0 ]]; then
    {
        while read lin; do          # for each object ask for the details
            echo "  $lin...c"
            if [[ $1 = "ManagedNode" ]]; then
                wmannode $lin > t1 2>&1 # details for the ManagedNode
            elif [[ $1 = "PcManagedNode" ]]; then
                wpcmngnode $lin > t1 2>&1 # details for the PcManagedNode
            else
                echo "*** ERROR - the resource type $1 cannot be managed"
                exit
            fi
            if [[ $? = 0 ]]; then
                awk -f add2pfm.awk "resource=$1" "node=$lin" "subs=$OSTot" t1
            else
                echo "*** ERROR - $1 $lin does not exist"
                exit
            fi
            rm t1
            echo "added"
        done
    } < t
    #refresh
    else
        echo "*** ERROR - Resource Type Unknown"
        exit
    fi
fi
rm t

```

---

## 9.6 Summary

This chapter included a grab-bag of hints, tips and helpful shell scripts that you may find useful when implementing a TME 10 environment.



---

## Chapter 10. In Conclusion

We hope you have found the information presented here useful and that it will help you plan for and successfully implement a TME 10 environment based on Version 3.2 of the TME 10 Framework. Obviously, what drives TME 10 installations are applications. As we write this, projects are in progress to create redbooks to address the LCF implementations of many of your favorite applications.

Keep an eye on the following web sites for the latest information related to LCF and upcoming redbooks related to Tivoli.

- <http://www.tivoli.com>
- <http://www.redbooks.ibm.com/solutions/tivoli>

Tivoli. The Power to Manage. Anything. Anywhere.





---

## Appendix A. Special Notices

This publication is intended to help customers and support personnel understand the new Lightweight Client Framework introduced in Version 3.2 of the TME 10 Framework. After reading this redbook, you should be in a position to plan for and implement a TME 10 environment based on LCF. The information in this publication is not intended as the specification of any programming interfaces that are provided by Tivoli's TME 10 Framework. See the PUBLICATIONS section of the IBM Programming Announcement for TME 10 Framework Version 3.2 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.

Licenses 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 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 been

reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX®	AS/400®
BookManager®	DB2®
IBM®	OS/2®
OS/390	RS/6000
SP1®	SP2®
System/390®	

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

---

## Appendix B. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

---

### B.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see "How to Get ITSO Redbooks" on page 157.

- *Understanding Tivoli's TME 3.0 and TME 10*, SG24-4948
- *TME 10 Cookbook for AIX*, SG24-4867
- *Setting Up a TME 3.0 NT Environment*, SG24-4819
- *A First Look at TME 10 Distributed Monitoring 3.5*, SG24-2112
- *Getting Started With TME 10 User Administration*, SG24-2015

---

### B.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
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
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

---

### B.3 Other Publications

The following publications shipped with the Version 3.2 of the TME 10 Framework are also important sources of further information:

- *TME 10 Framework Planning and Installation Guide*
- *TME 10 Framework User's Guide*
- *TME 10 Framework Reference Manual*
- *TME 10 Framework Release Notes*



---

## 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 URL <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:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

To get LIST3820s of redbooks, type one of the following commands:

```
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:

```
TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
```

To register for information on workshops, residencies, and redbooks:

```
TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
```

For a list of product area specialists in the ITSO:

```
TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
```

- **Redbooks Home Page on the World Wide Web**  
<http://w3.itso.ibm.com/redbooks>
- **IBM Direct Publications Catalog on the World Wide Web**  
<http://www.elink.ibm.link.ibm.com/pb1/pb1>

IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to [announce@webster.ibm.link.ibm.com](mailto:announce@webster.ibm.link.ibm.com) with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

---

## 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** (Do not send credit card information over the Internet) — send orders to:

	<b>IBMMAIL</b>	<b>Internet</b>
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 Publications Customer Support P.O. Box 29570 Raleigh, NC 27626-0570 USA	IBM Publications 144-4th Avenue, S.W. Calgary, Alberta T2P 3N5 Canada	IBM Direct Services Sortemosevej 21 DK-3450 Allerød Denmark
--	--	--

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States) or (+1)001-408-256-5422 (Outside USA)** — ask for:

Index # 4421 Abstracts of new redbooks  
Index # 4422 IBM redbooks  
Index # 4420 Redbooks for last six months

- **Direct Services** - send note to [softwareshop@vnet.ibm.com](mailto:softwareshop@vnet.ibm.com)

- **On the World Wide Web**

Redbooks Home Page	<a href="http://www.redbooks.ibm.com">http://www.redbooks.ibm.com</a>
IBM Direct Publications Catalog	<a href="http://www.elink.ibm.link.ibm.com/pbl/pbl">http://www.elink.ibm.link.ibm.com/pbl/pbl</a>

- **Internet Listserver**

With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to [announce@webster.ibm.link.ibm.com](mailto:announce@webster.ibm.link.ibm.com) with the keyword subscribe in the body of the note (leave the subject line blank).

---

## 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.**

**DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.**





---

## Index

### A

ADR marshalling functions 120  
after\_install\_policy 36, 64, 68  
allow\_install\_policy 35, 64  
application development 107

### B

backup 125  
bibliography 155  
Bourne shell 29  
broadcast 34, 36, 49, 64, 90

### C

client component 112  
client considerations 30  
command line interface 8, 18  
commands  
    waddrealm 77  
    wadminep 19  
    wbkupdb 90, 126  
    wchdep 111  
    wcpdrom 129  
    wcrigate 19, 44  
    wdebug 115  
    wdelep 19  
    wdelgate 19  
    wdelrealm 77  
    wdepset 111  
    wep 19, 86  
    wgateway 19, 44  
    wgetepol 19, 71  
    winstlcf 18, 48, 55  
    wlsrealms 77  
    wpatch 103  
    wpreinst.sh 129  
    wputepol 19, 71  
    wserver 41  
    wsetpm 18  
    wstarthttpd 77  
    wstophttpd 77  
    wupdefhtml 78

common porting layer 121  
configuration 28  
configuration LCF 61  
Configuration Settings Page 83  
connection process 34  
CORBA 1, 3  
core management functions 8  
csh 29

### D

data transfer 5  
data types 121  
database 6, 20, 90, 125  
DDNS 32  
DefaultHTMLItems 78  
Desktop 9, 17, 23, 27, 41, 44  
DHCP 32  
distributed exceptions 117  
DNS 31  
downcall 14, 108, 110, 112  
DynaText 21

### E

Endpoint 13, 24, 28, 47  
    connection process 34  
    Gateway 13, 14, 24, 27, 29  
        installation 43  
    installation 47  
    login 61  
    Manager 13, 15, 23  
        installation 41  
        policy 16, 19, 45, 49  
        web server 78  
Endpoint method 108  
Enterprise Console 20  
etc/hosts file 32

### F

file system input/output 119

file0.tar 129  
framework  
    database 6, 90, 125

## G

Gateway 27, 29  
Gateway method 108  
gethostbyaddr 31  
gethostbyname 31

## H

hardware requirements 25  
HTTP daemon 22, 75

## I

Implementation Repository 4  
installation 8, 41  
InstallShield 48  
interconnection 11, 33  
Interface Repository 4  
Inventory 20  
IPX 10, 24

## K

keyboard 31

## L

last.cfg 36, 59, 62  
LCF 1, 12  
LCF daemon 91  
LCF Daemon Status Page 79  
LCF object 108  
LCF programming environment 114  
lcf\_oserv\_upgrade 103  
lcf.dat 62  
lcf.dat file 36  
lcf.id 62  
lcf daemon 13, 19, 28, 34, 37, 115  
lcf.d.cfg 62  
lcf.d.log 56, 62  
lcs.login\_interfaces 37

libcpl 115, 121  
libdes 115  
libmrt 115  
Lightweight Client Framework  
    *See* LCF  
limitations 11  
limiting broadcasts 36  
Logfile Page 81  
logging functions 119  
login\_policy 36, 64, 70  
logon script 53  
ltid 115

## M

maintenance 91  
Managed Node 10, 23, 27, 29  
MDist repeater 15  
memory management 116  
Method Cache Page 82  
mid-level manager 112  
migration 89

## N

name resolution 29  
Netware 10  
Network Address Configuration Page 85  
network requirements 31  
NIS 32  
nobody ID 29  
nslookup 29

## O

Object adapters 4  
Object Management Group  
    *See* OMG  
odadmin 31  
odstat 126  
OMG 1, 3  
one-way connection 33  
operating system independence 2  
operating systems 24  
ORB 3, 35

## P

PC Managed Node 10, 24, 28  
ping 29  
planning 23  
policy 35, 45, 49, 61, 64, 71, 90  
policy regions 18  
process slots 26  
profile 18, 35, 68, 92

## R

RDBMS Interface Module 20  
recovery 125  
relational databases 20  
removing Managed Nodes 128  
resolver 31  
reverse name resolution 29  
RIM 20  
runtime library 115

## S

scalability 38, 110  
security 91  
select\_gateway\_policy 35, 64, 66  
sequence manipulations 118  
spawner 13, 14, 19, 28, 34, 37, 91, 109, 115  
subscriber 18, 68, 92  
SunOS 29  
supported operating systems 2  
swap space 25

## T

TEIDL 20, 114  
throughput 31  
tips 125  
Tivoli Management Region  
    *See* TMR  
Tivoli\_Admin\_Priveleges 30  
TME 10 Desktop 9, 17, 23, 27, 41, 44  
TME methods 107  
tmersrvd 30  
tmeservd 28  
TMR 6, 9, 33

TMR interconnection 11  
TMR Server 10, 23  
top-level manager 112  
topology 17  
Trace Log Page 84  
TRIP 30  
two-way connection 34

## U

upcall 15, 108, 110, 113  
upcall collector 112  
Usage Statistics Page 82

## W

waddrealm 77  
wadminep 19  
wbkupdb 90, 126  
wchdep 111  
wpcdrom 129  
wrtgate 19, 44  
wdebug 115  
wdelep 19  
wdelgate 19  
wdelrealm 77  
wdepset 111  
web interface 22, 75  
wep 19, 86  
wgateway 19, 44  
wgetepol 19, 71  
winstlcf 18, 48, 55  
wlsrealms 77  
wpatch 103  
wpreinst.sh 129  
wputepol 19, 71  
wserver 41  
wsetpm 18  
wstarthttpd 77  
wstophttpd 77  
wupddefhtml 78



---

## ITSO Redbook Evaluation

TME 10 Framework Version 3.2: An Introduction to the Lightweight Client Framework  
SG24-2025-00

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.com>
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to [redeval@vnet.ibm.com](mailto:redeval@vnet.ibm.com)

**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! )

---

---

---

---

---



Printed in U.S.A.

SG24-2025-00

