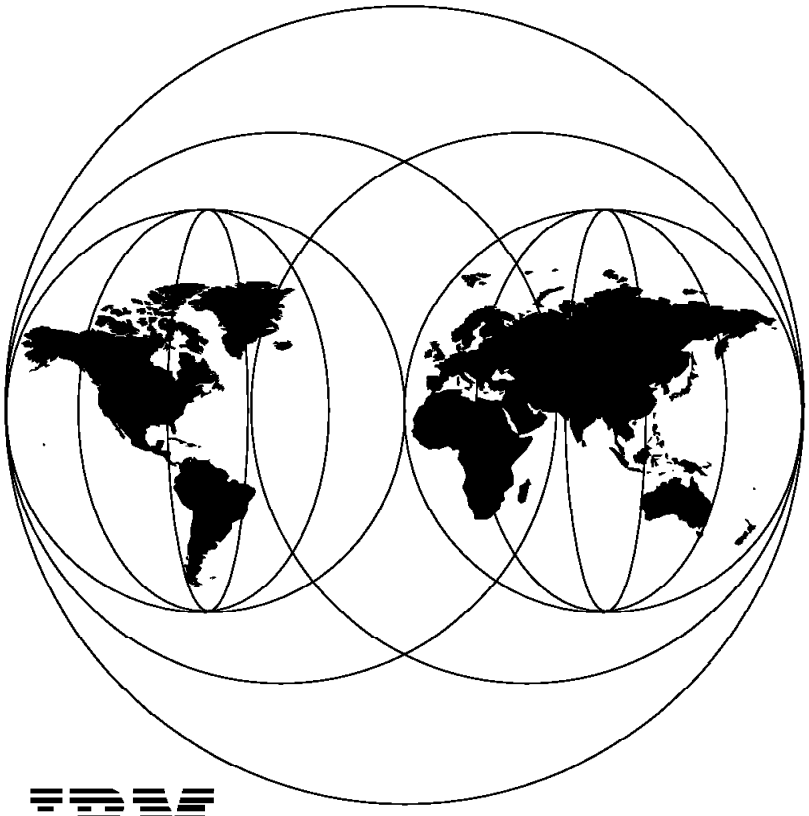
SG24-2112-00

A First Look at TME 10 Distributed Monitoring 3.5

July 1997



IBM
International Technical Support Organization
Raleigh Center

International Technical Support Organization

SG24-2112-00

A First Look at TME 10 Distributed Monitoring 3.5

July 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 83.

First Edition (July 1997)

This edition applies to TME 10 Distributed Monitoring Version 3.5.

Warning

This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. It is recommended that, when the product becomes generally available, you destroy all copies of this version of the book that you have in your possession.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 678
P.O.Box 12195
Research Triangle Park, NC 27709-2195

Contents

Figures

Preface

This redbook introduces the latest version of TME 10 Distributed Monitoring, previously known as Tivoli/Sentry. It describes the operation of the product in the current Tivoli Management Environment and also in the new Lightweight Client Framework (LCF) model.

This redbook will help you position TME 10 Distributed Monitoring and TME 10 as a solution for enterprise systems management and show practical examples for its use.

If you are planning to install TME 10 Distributed Monitoring, or if you already have it installed, this book will help you to understand the new functions in Version 3.5. If you want to learn about the enhanced scalability provided by the TME Lightweight Client Framework, this book will give you a technical overview illustrated by practical examples of how LCF is exploited by TME 10 Distributed Monitoring 3.5.

Some of the new features of TME 10 Distributed Monitoring covered in this redbook include database monitoring (Oracle) and data collection and graphing using the new TME 10 Distributed Monitoring World Wide Web (WWW) interface.

The scenarios are documented in a way that service providers can use the examples as a base in client implementations.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the Systems Management and Networking ITSO Center, Raleigh.

Rob Macgregor is a technical specialist at the Systems Management and Networking ITSO Center, Raleigh. He writes extensively and teaches IBM classes worldwide on systems management and network security. Before joining the ITSO three years ago, Rob worked in the UK technical support center, dealing with network and systems management products.

Stefan Uelpenich is an Advisory ITSO Representative, working as a project leader at the Systems Management and Networking ITSO Center, Raleigh. He writes extensively and teaches IBM classes worldwide on all areas of systems management. Before joining the ITSO, he worked in IBM Germany′s Professional Services organization as an Advisory I/T Architect for Systems Management, consulting major IBM customers.

Andreas Kuffer is a systems management specialist in IBM Germany. He has three years of industry experience in the area of UNIX and networking. His areas of expertise include UNIX system administration and open network management.

Peter Glasmacher is a Consultant in IBM Germany. He has about 15 years of experience in networking, focusing on systems management for the last seven years. He has worked at IBM for 24 years. His areas of expertise include network design/implementation, security consulting and design/implementation in the Systems Management arena.

Graeme Naysmith is an Advisory I/T Specialist working in Warwick, England. He joined IBM in 1985 and has four years AIX/UNIX monitoring and automation experience. His areas of expertise include network and systems management. Graeme is currently involved in the migration of SystemView applications to the TME 10 product set.

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

David Boone, Linda Robinson, Shawn Walsh, Gail Doucette Wojton and Paul Braun
Systems Management and Networing ITSO Center, Raleigh.

Greg Kattawar, Astrid Burnette, Sean Starke and Carol Corley
Tivoli Systems, Austin.

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 93 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`

- Send us a note at the following address:

  `redbook@vnet.ibm.com`

Chapter 1. Introduction

Under the Tivoli Management Environment, the process of managing distributed systems is divided into a number of application categories:

Applications This area covers the management of standard application environments, such as Lotus Notes or Internet servers, and also the process of making any application *management-ready* so that it is instrumented and controllable.

Availability This area covers the monitoring of distributed systems and networks. It also covers centralized monitoring of events and performance characteristics.

Deployment This area deals with the distribution and implementation of applications on distributed systems.

Operations and Administration This area covers remote control of workstations and scheduling of background jobs in a distributed environment.

User and Security This area covers all aspects of administrating security in a distributed environment, creation and maintenance of user accounts and administration of access policies.

All of these management applications are based on a consistent framework, the Tivoli Management Platform (TMP) which provides them with services for remote execution, authentication, data distribution and transaction control.

This book deals primarily with an application that falls into the Availability category: *TME 10 Distributed Monitoring*. It describes the functions available in the latest version of TME 10 Distributed Monitoring, Version 3.5.

1.1 About TME 10 Distributed Monitoring 3.5

TME 10 Distributed Monitoring is an application that provides monitoring of status and performance information for remote systems from a central point. It makes extensive use of the TME platform for distribution of monitoring policy, alerting, and automation.

Prior to Version 3.5, TME 10 Distributed Monitoring provided a wide range of monitors for UNIX and NT systems and a proxy monitoring option for management of non-platform operating systems and SNMP resources. We discuss the way that TME 10 Distributed Monitoring works and how it fits into the TME framework in Chapter 2, “TME 10 Distributed Monitoring” on page 3.

Version 3.5 introduces a number of enhancements to the existing product:

- Support for NetWare servers, using the new TME Lightweight Client Framework platform extension.
- Support for historical data collection and a Web-based monitoring facility for viewing collected data.

- A number of additional collections of monitors, for Oracle, Sybase and the TME 10 Enterprise Console.

We discuss all of the enhancements, with examples, later in this book.

1.2 About the Lightweight Client Framework

One of the most significant features of TME 10 Distributed Monitoring 3.5 is that it is the first TME application to provide support for Lightweight Client Framework (LCF) endpoints. LCF has a number of objectives:

- To improve the scalability of the Tivoli Management Environment, by removing some of the design limitations in the full platform
- To extend the full function of the TME platform to a large number of desktop managed nodes
- To minimize the administrative cost of this increased scalability and scope
- To achieve all of the above with a minimal impact on the managed system

We describe how LCF works in Chapter 3, “Introducing the Lightweight Client Framework” on page 13 and we show examples of implementing it on two different operating systems in Chapter 4, “Setting Up the Lightweight Client Framework” on page 21.

Chapter 2. TME 10 Distributed Monitoring

In this chapter we describe the detailed operations of TME 10 Distributed Monitoring. If you are familiar with the product, you may wish to skip this section.

TME 10 Distributed Monitoring, previously known as Tivoli/Sentry, provides monitoring functions for a range of UNIX platforms plus Windows NT. In the latest release (TME 10 Distributed Monitoring 3.5) it also supports Novell NetWare R3 and R4 servers. Sentry is based on the Tivoli Management Environment and it uses TME functions for many operations, such as deploying monitors to distributed systems, defining monitoring policies and sending events.

To have a good understanding of any TME application you need to understand the function provided by the TME platform. We briefly introduce the platform here, but for a more detailed treatment you may want to refer to *TME 10 Cookbook for AIX*, SG24-4867 or *Understanding Tivoli's TME 3.0 and TME 10*, SG24-4948.

2.1 Introduction to The Tivoli Management Environment

The Tivoli Management Environment (TME) contains a distributed object-oriented infrastructure, a set of tightly integrated systems management applications and a set of interfaces. The interfaces include a GUI for simple control, an extensive command line interface that makes it easy to perform batch operations and automation and a set of APIs to allow other applications to be integrated into the framework.

Figure 1 shows a schematic view of TME.

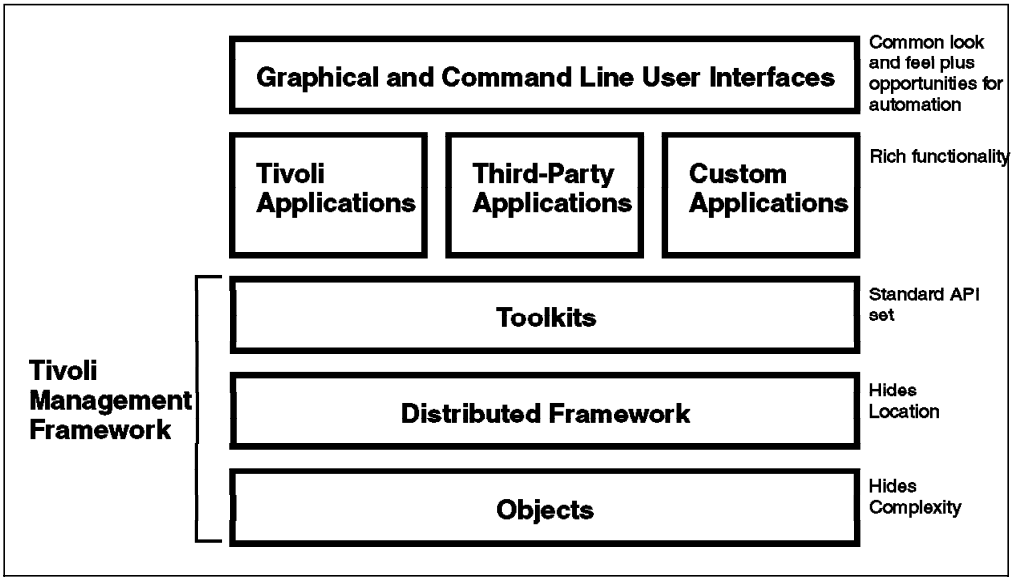


Figure 1. TME Architecture

The Tivoli Management Framework, also referred to as the *platform* is fundamental to everything that a TME application does. It is a distributed object request broker (ORB) implementation, based on the common object request broker (CORBA) standard. An object request broker is a function that allows

object-oriented programming techniques to be applied in a distributed environment. The benefit of an object-oriented approach for heterogeneous systems management is that the real world configuration can be masked. An application just needs to know the interface definition of an object (the data it exposes and the methods it provides). The problems of implementing the method on NT or Solaris or AIX or whatever, are encapsulated within the object itself.

In addition to the base ORB functions, the TME framework provides a number of other services, for example:

- Administration functions
- Security (authentication and access control)
- Transaction control
- Packaging and distribution functions

It is these capabilities that make TME a specialized systems management environment. As we go on to describe the operation of Sentry you will see how the platform functions simplify the creation of an application.

2.1.1 Tivoli Management Regions

We have said that TME provides a distributed framework for systems management applications, but how does that look in practice? Systems within the framework are placed within *Tivoli Management Regions* (TMRs). A TMR is comprised of one server system and a number of clients, or *managed nodes*. Each system runs an object request broker (the oserv daemon) and an application on one system within the TMR can invoke a function on any other system by using the ORB services.

For many operations, all the systems in the TMR are peers to each other. Why, then, is one designated the TMR server? The answer is that certain key functions require a single point of reference, which the server provides. The main functions provided by the server are object location services and security controls. Some applications also use the server as a default location if they have data or code that does not need to be present in all managed nodes.

Practical considerations limit the number of managed nodes within a TMR to about 200. To manage a larger population of nodes, TMRs have to be interconnected. The TMR server is responsible for communicating with the server in an adjacent TMR.

The TME platform is a sizeable piece of code, which may not be desirable or possible to run on every system you want to manage. To deal with this case, you can use *PC managed nodes*. A PC managed node is a small piece of code that runs on NetWare, OS/2, and all types of Windows systems. It implements a simple endpoint function for the most commonly required TME applications, software distribution and in certain cases user administration.

Although the PC managed node meets the basic requirements for desktop systems, it is not very flexible and it introduces some problems of security and maintainability. The answer, which will appear throughout the Tivoli product range during 1997, is the *Lightweight Client Framework* (LCF). This provides support for many of the Tivoli APIs, but does so with a minimal disk footprint and

without any installation or pre-definition required. Figure 2 on page 5 shows how TMRs are put together.

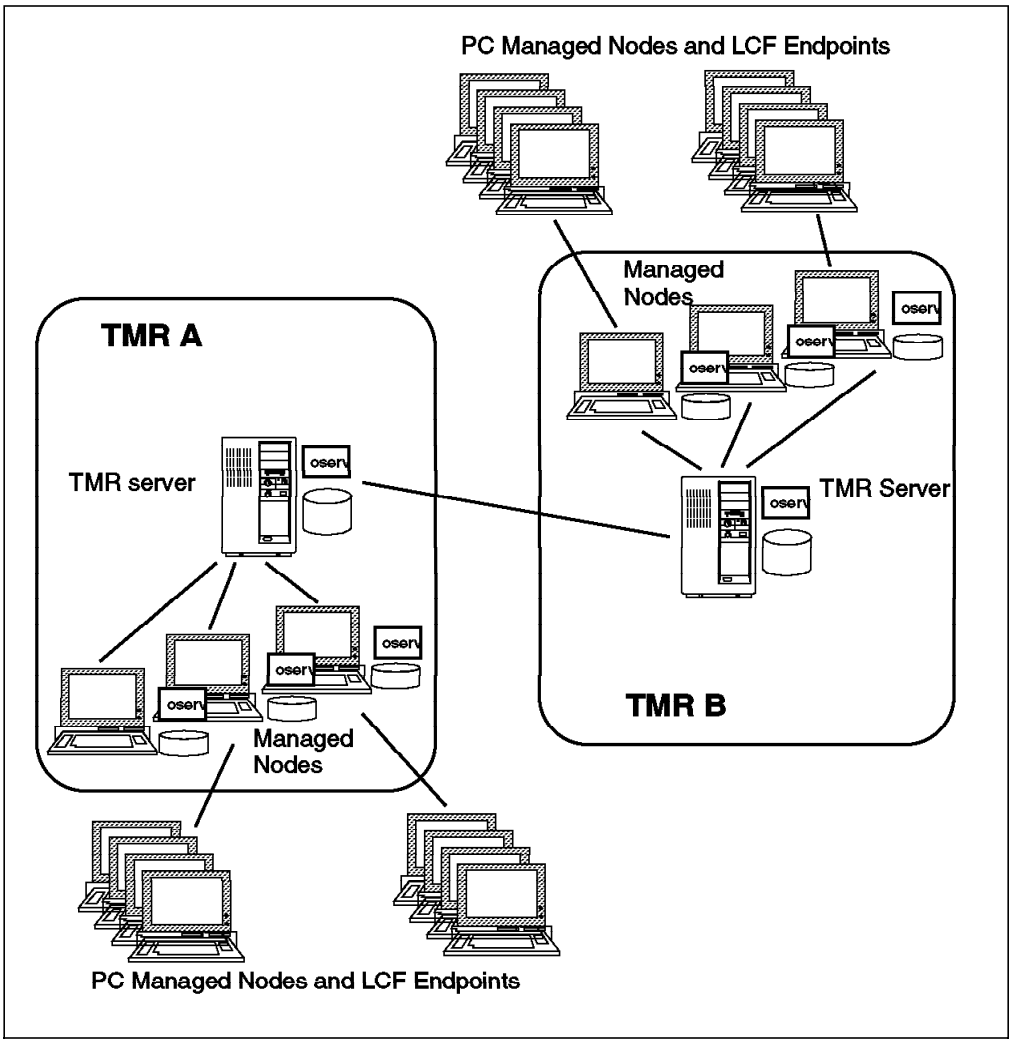


Figure 2. TMR Components

2.1.2 Administrators and Policy Regions

If you want to use any of the Tivoli applications you need an administrator ID. This ID is *not* a regular system ID, although TME uses a standard system user ID and password to provide authentication. In other words, you prove your credentials by providing your system ID and password. TME then maps that ID onto an administrator ID. What this means is that you may have root access to one of the systems in the TMR, but that does not necessarily give you any TME authorization. Conversely, you can perform tasks that would normally need root access using a regular personal user ID, if your TME access permits it.

Access controls in TME are very granular. Every object (whether a real resource, such as a managed node or a logical entity, or a file package or a system monitor) is created within a *policy region*. Each administrator holds specific authorization roles within the policy region. So for example, an administrator may have authority to update Sentry monitors on one group of systems, but not on another.

Authorization roles are also applied at the TMR level. If you have a particular role in the TMR, it overrides your authorization at the policy region level.

2.1.3 Management By Subscription

In TME, all application configurations are distributed based on policy, rather than by configuring nodes individually. The way this works is a concept called *management by subscription*. All application functions are configured using profiles in the oserv database. The contents of a profile depends on the application that created it. For example, a TME 10 Distributed Monitoring profile would contain system monitoring details, whereas a TME 10 Software Distribution profile would contain file package descriptions. Profiles are contained in objects called *profile managers*. Nodes can be subscribed to these profile managers and, when a profile is distributed, it is applied to all of the subscribed nodes.

You can also create hierarchies of profile managers, by subscribing one profile manager to another. This provides a very flexible way to have centralized management for some elements of a system and distributed management for others. Figure 3 shows an example of an organization that has a number of workstations. There are some Sentry monitors that are applied to all of the workstations. However, some parts of the organization have specific monitoring requirements. The Finance department, for example, uses an RDBMS, so they want monitors to check that it is running correctly. Over in the Research department they run memory intensive analysis, so they need to keep an eye on swapping activity. The hierarchy in the diagram allows this environment to be created with minimum effort, each monitor and each node subscription being defined only once.

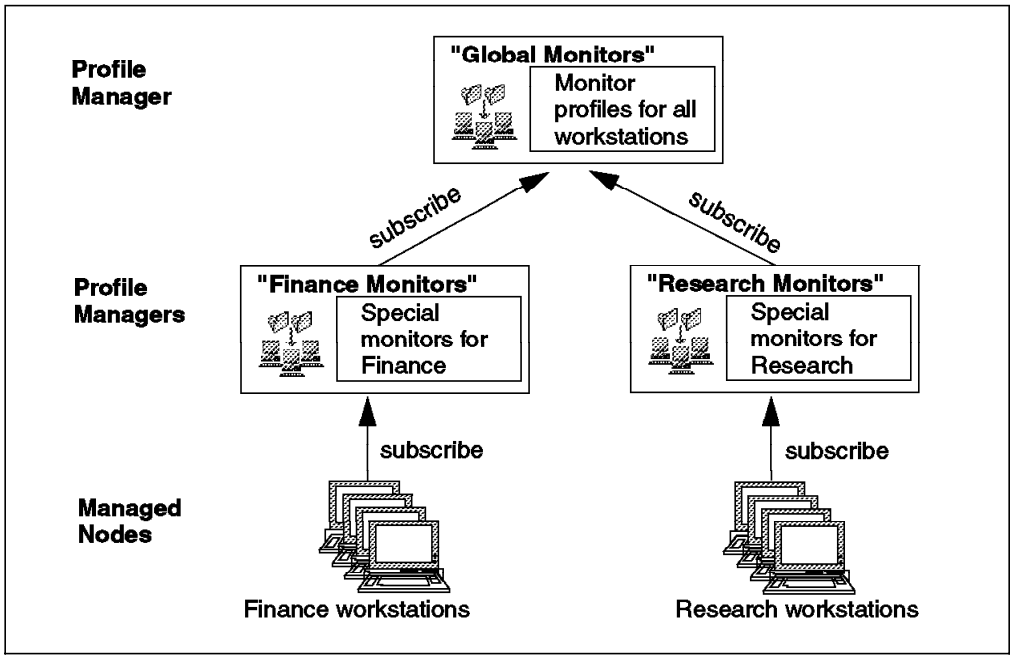


Figure 3. Hierarchy of Profile Managers

2.2 Inside TME 10 Distributed Monitoring

TME 10 Distributed Monitoring gives you the capability to create monitors for many different aspects of a TME managed node. It is installed like any other TME application, by selecting **Desktop**, **Install** and then **Product** from the TME desktop or using the winstall command. This installs a number of components. The main ones are:

- The Sentry engine. This is the process that actually performs the monitoring. All monitors are executed locally on the managed node (except for the special case of proxy monitors). The Sentry engine is a timer-driven program that runs as a background task (as a daemon in the case of UNIX, as a service on NT). Every minute it wakes up and processes any monitors that are queued for execution at that time.
- Endpoint Classes. These are object classes and methods that are installed and run on the managed node and are responsible for updating the Sentry engine whenever monitor changes are distributed.
- Kennel Classes. These are object classes and methods that are installed on the server. They perform the defined actions when a Sentry monitor hits a threshold.

Each monitor is in the form of a command or program that returns a result. The monitor result is tested against a number of threshold levels, any of which can have actions associated with it. There are three error threshold levels named *warning*, *severe*, and *critical* in order of increasing scariness. There is one other level defined, *normal* which is to trigger an action when the result does not indicate a problem. Finally, there is a threshold level named *always* that is triggered regardless of the test result. It would be possible for a single monitor to include definitions for all of these threshold levels, but normally only one or two of them are appropriate. In these cases the other levels can be left undefined.

When a monitor triggers a threshold there are a number of actions available:

- An administrator can be notified, via a TME notice, or pop-up message.
- An indicator can be set on the TME desktop.
- An event can be sent to the TME 10 Enterprise Console.
- Automated action can be taken, either a program or a TME task.

2.2.1.1 Monitoring Collections

Sentry can monitor UNIX, NT and NetWare systems. Although the Tivoli Management Environment provides a consistent way to invoke monitoring, regardless of the system platform, the monitors themselves differ from one platform to another. Sentry reflects these variations by placing monitors into groups called *monitoring collections*. Each collection contains a group of related monitor definitions. You have to install the collections you need, dependent on the types of system you want to manage and the applications they are running.

The monitoring collections are defined in the oserv database on the server only. When a monitor is distributed the code appropriate to the particular platform is sent to the managed node for processing.

2.3 Under the Covers of TME 10 Distributed Monitoring

Let us now look in some more detail at the way that Sentry operates. We divide the operation into two steps:

1. Creating a monitor
2. Executing a monitor

2.3.1 Creating a Monitor

Figure 4 shows how a monitor is instantiated on a managed node.

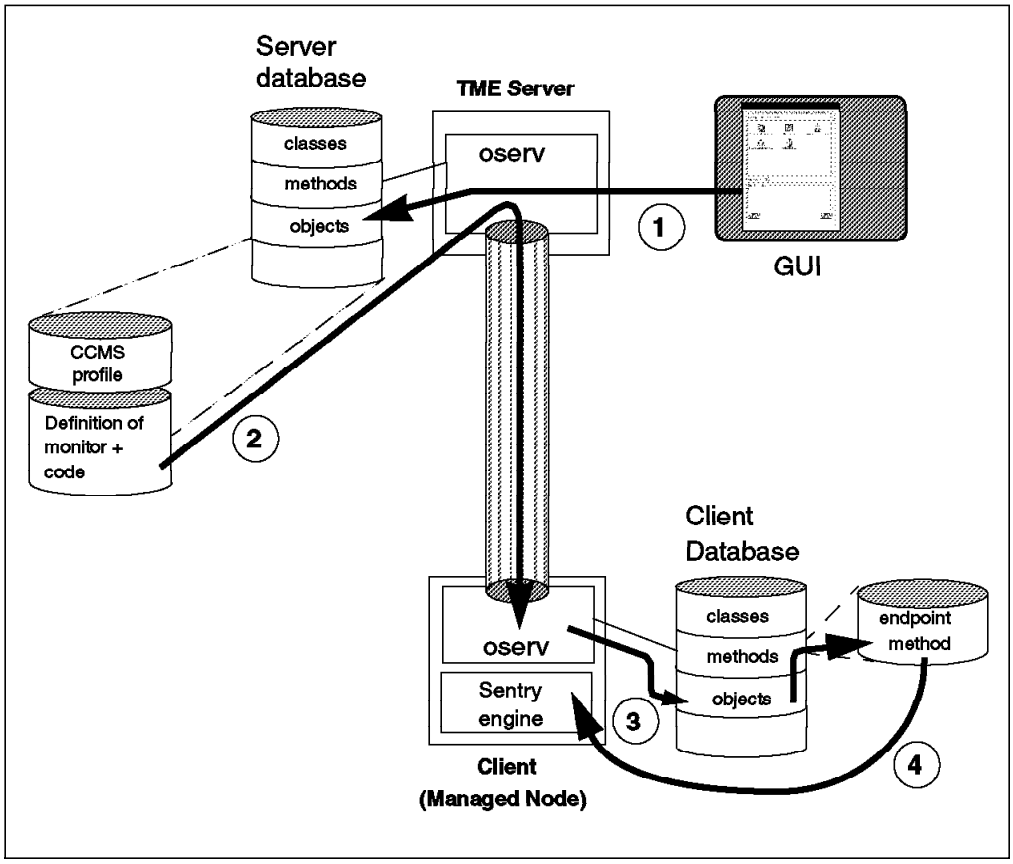


Figure 4. Creating a Monitor

1. The administrator creates a monitor profile using the GUI or the waddmon CLI command. The profile can contain multiple monitor definitions. Each monitor is actually an instance of an object class, stored in the server oserv database.
2. The administrator distributes the profile, either directly or by scheduling it as a background task. The TME platform provides facilities for creating and distributing Change Control Management System (CCMS) profiles, effectively a package of code and data containing the instructions for the new monitor. Because this is using a standard platform function, it can benefit from features such as mdist (multiplexed distribution) which will reduce network load and minimize repeated transmissions.
3. The distribution process kicks off the monitor installation by invoking a Sentry endpoint method on the target managed node.
4. The endpoint method updates the Sentry engine with the new monitor profile.

2.3.1.1 Executing a Monitor

Sentry monitors start to operate as soon as they are installed in the Sentry engine. The minimum polling interval is one minute, so the Sentry engine wakes up at one minute intervals and looks to see what monitors are waiting in that particular time slot. It is possible to overload the Sentry engine by giving it more monitors than can be executed within a one minute window. If this happens, it is possible that it will never catch up. The moral of this is: make sure that the monitors will run in a reasonable timescale, and do not set unrealistically short polling intervals.

When the monitor detects a result that triggers one of the defined response levels, it will cause an action to be taken. Some of these actions, such as local logging or local command execution, can be executed directly on the managed node. However, most actions involve invoking a method on another node in the TMR. This is where the kennel classes come into play.

Figure 5 shows what happens.

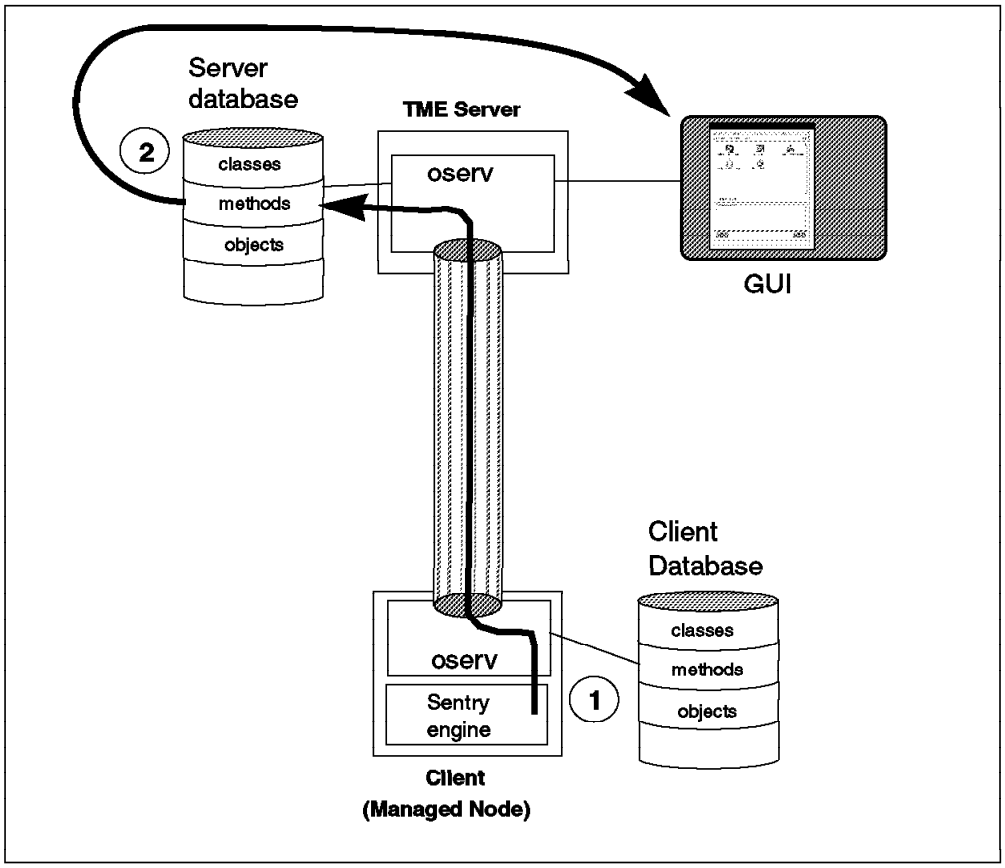


Figure 5. Executing a Monitor Action

1. The Sentry engine invokes a method call on the TMR server. Calls to a remote ORB contain an object reference and a method invocation. In this case the object is the monitor instance and the method is whatever action is defined in the monitor. The monitors are instantiated on the TMR server, so that is where the invocation is made.
2. The action method invokes the desired action(s), such as updating an indicator icon, running a task sending e-mail, etc. These actions are

themselves method invocations that can be invoked on whichever TME node is appropriate.

2.3.2 Displaying Events

There are several ways of displaying events using TME 10 Distributed Monitoring. Within each monitor you can define that events are sent to any combination of the following:

- The TME desktop
- A TME Notice group
- The Tivoli Enterprise Console (T/EC)
- A logfile
- Via a mail message

Events may be sent to any pre-defined adminstrator.

Because you can also execute TME tasks or programs as a result of a monitor being triggered, you can also send events to other event handlers.

2.3.2.1 Generating Pop-Up Messages

Pop-ups can be defined to appear on any TME administrator's desktop, as shown in Figure 6.

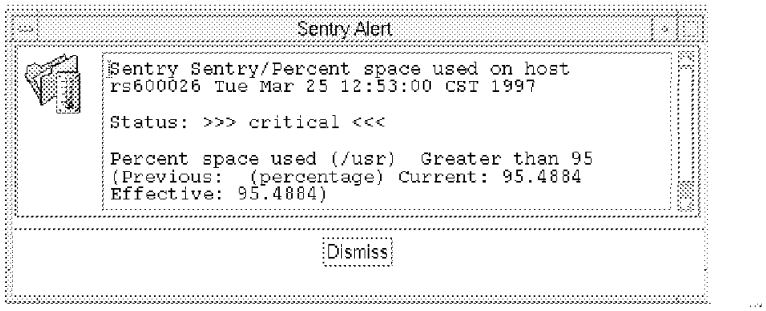


Figure 6. An Event Pop-up Message

2.3.2.2 Notice Groups

Notice groups are a TME facility for storing, organizing and presenting event information. Sentry has the capability to send alerts to specific notice groups. These are:

- Sentry
- Sentry-log
- Sentry-urgent
- SentryStatus

When you define a TME administrator, you can specify which notice groups they will be subscribed to. An event sent to a notice group can be read when convenient and will not produce a popup, although the Notices icon changes to show that unread notices have arrived (see Figure 7 on page 11).

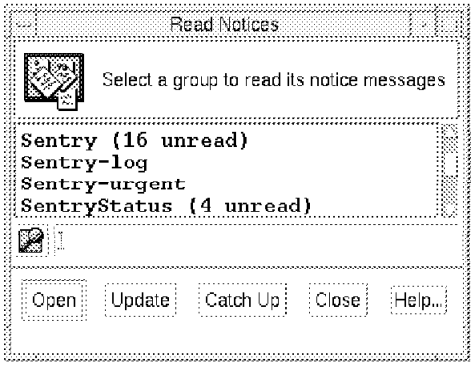


Figure 7. Sentry Notices

2.3.2.3 The TME 10 Enterprise Console (T/EC)

The pop-up and notices facilities are a standard part of the TME platform. T/EC, on the other hand, is an additional TME application specifically designed for handling event data. T/EC uses an RDBMS to organize events and it is built around a powerful rules engine which allows events to be correlated regardless of generation time and source. These rules provide filtering of potentially high numbers of events (see Figure 8).

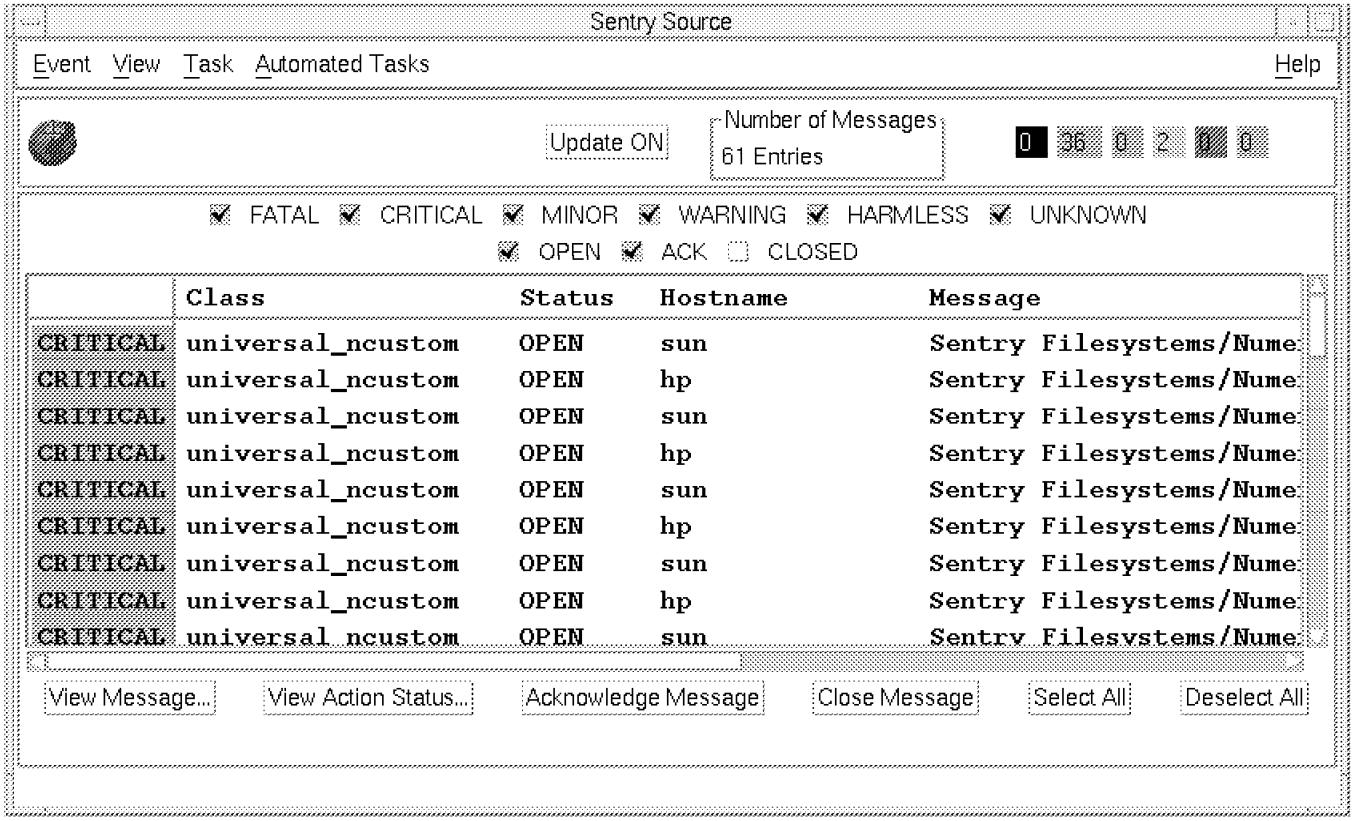


Figure 8. Events Sent to the T/EC

Chapter 3. Introducing the Lightweight Client Framework

TME 10 Distributed Monitoring 3.5 is the first TME appplication to exploit the new Lightweight Client Framework (LCF). In this chapter we describe the factors that led to the development of LCF and explain in detail how it works.

3.1 Stresses and Strains in the TME Framework

There is no theoretical limit to the number of managed nodes that can be handled within the TME framework. As your environment grows you can go on interconnecting TMRs and distributing application function through the framework. However, as we have mentioned, there are a number of problems with this model:

- The environment becomes increasingly complex. TMRs have to be adjacent to allow full control across the boundaries. This leads, potentially, to a great many inter-TMR connections and the development of single points of failure. The ORB databases on the connected servers and clients have to be synchronized. Apart from the possibility of database corruption arising as the number of nodes grows, it becomes increasingly difficult to effectively back up the databases.
- The platform and applications use a considerable amount of system resource, principally disk space. Furthermore, this code base needs to be maintained. TME code is installed using standard platform functions, but it is only distributed within a TMR (unlike normal software distribution, which can span TMR boundaries). Keeping all systems in line can become a serious headache.
- Although the platform is implemented on many server operating systems, it does not support the most common desktop systems, Windows 3.1, Windows 95 and OS/2.

3.1.1 PC Managed Nodes

PC managed nodes extend TME support to these desktop systems, providing endpoint support for the most common management requirements, software distribution and task execution. The PC managed node software is small in size and reliable. It removes the 200 nodes per TMR restriction of managed nodes, allowing you to create TMRs supporting thousands of nodes, in a ratio that matches the real life ratio between server and client systems.

Introducing PC managed nodes also reduces the load on the TMR server. The main way it achieves this is by introducing Mdist repeaters into the network. Mdist was first introduced between TMR servers, so that data destined for a number of machines in an adjacent TMR is only sent once over the connection and then fans out on the other side. With PC managed nodes the principal is extended so that managed nodes within the same TMR can act as Mdist repeaters for a group of desktop systems.

PC managed nodes greatly improve the scalability and managability of the TME architecture, but there are still a number of problems associated with them:

- Maintenance is an issue. The PC managed node software contains application endpoint functions. As changes are made to the applications, or a 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 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.

3.2 LCF to the Rescue

The Lightweight Client Framework (LCF) retains the best features of a PC managed node, such as a small system footprint and Mdist data distribution. However it also provides many of the services provided by the full platform. The way LCF achieves this is by implementing a *dataless endpoint*.

LCF will be implemented across all Tivoli supported platforms. When all of the TME applications support LCF endpoint nodes, the design criteria for TMRs will be fundamentally altered. We can envision TMRs containing a small number of managed nodes, which will be dedicated systems management machines. Surrounding these will be all of the other systems, which currently are either managed nodes or PC managed nodes, but which can now be LCF endpoints. Because there will need to be fewer managed nodes, there will be fewer reasons to design interconnected TMRs, leading to much improved simplicity and reliability.

3.2.1 Components of LCF

There are three roles in the LCF architecture:

1. The *endpoint*. This is the LCF node itself. It runs a small daemon (background) function, called the *spawner*. The spawner program is called lcfd. This is responsible for executing program calls (actually, method invocations) from other TME nodes. Its only connection to and knowledge of the rest of the TME world is through an endpoint gateway.

   You do not have to install any other code on the endpoint, because any application code is downloaded automatically as needed.

2. The *endpoint gateway*. This is a managed node (usually a client, but it could be the TMR server). An endpoint gateway may have a number of LCF endpoints assigned to it. Its job is to pass data and method invocations to the spawner to execute (known as *downcalls*) and handle method requests from the spawner (*upcalls*). Endpoint gateways are automatically defined as Mdist repeaters for all of the endpoints they serve. This gives you the benefit of an intelligent distribution mechanism without any administrative overhead.

3. The *endpoint manager*. This is the function that allocates endpoints to endpoint gateways. Currently the endpoint manager has to also be the TMR server, but future implementations may remove that requirement. Policy methods are available to control the way that endpoints are allocated to gateways.

Figure 9 on page 15 illustrates the relationships between the three roles in LCF.

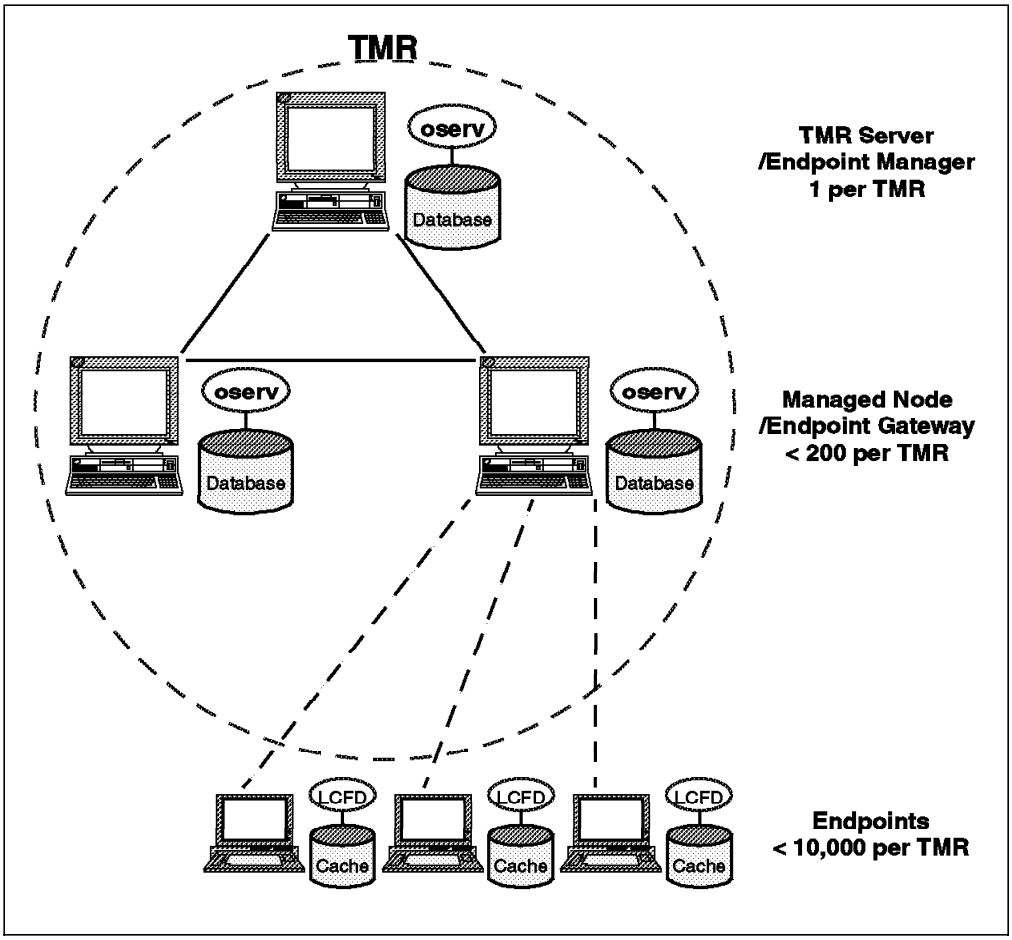


Figure 9. Components of LCF

3.2.2 How an Endpoint Gets Connected

The base LCF code has to be installed on the endpoint manually, using normal system facilities (using *setup* and the InstallShield function on Windows systems for example). However, the only function that has to be installed is the spawner and a few local utility commands, so this is a very small installation; typically less than 1 MB. Tivoli Systems has entered into a preliminary agreement with Intel Corporation to have the LCF spawner pre-installed in read-only memory on new PC motherboards.

The spawner does not need any intitial customization, so when it first starts it does not have an assigned endpoint gateway. To get connected to a gateway, the endpoint initiates the sequence shown in Figure 10 on page 16.

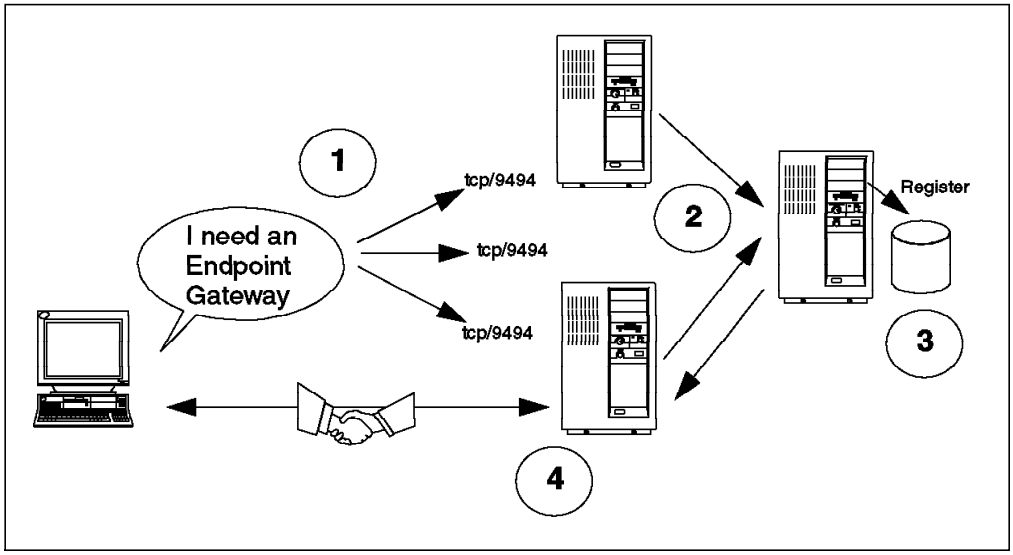


Figure 10. Initial Endpoint Connection

The numbered steps are as follows:

1. The endpoint broadcasts on TCP port 9494, asking to be connected. All of the gateways listen on this port, so any active 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 their TMR server, in the present version). This is sent using the normal ORB (oserv-oserv) communications channel.
3. The endpoint manager registers the new endpoint in the oserv database and assigns it to an endpoint gateway. A number of policy methods are provided to allow you to control how a gateway is chosen and execute any other automatic function. For example, you may want to subscribe the new resource to profile managers, or alert someone that it has been connected.
4. The chosen gateway responds to the endpoint connection request. Gateway and endpoint perform an initial handshake to establish their identities and generate encryption keys for authentication purposes.

3.2.3 LCF Methods

A major objective in the LCF design was to make it appear as similar as possible to the full framework, from the point of view of an application. This means that many of the CORBA programming interfaces are retained. Some differences inevitably arise, however, because of the nature of the endpoint environment. For example:

- An endpoint has no object database, so each transaction has to instantiate objects as they are needed.
- The endpoint has no notion of other TME nodes beyond its endpoint gateway. This means that applications have to be written to handle endpoint requests at the gateway, instead of at the target node.

Another major objective of the LCF design was to improve the scalability of the TME framework. The smart approach to increasing scalability is to reduce the cost of each transaction, allowing more systems to be managed by less dedicated resource. In particular, the TMR server can be a bottleneck as he

number of managed resources increases, so anything that reduces load on it is very beneficial.

We now look in detail at how LCF changes the way that applications operate in TME. There are two parts to this.

1. Endpoint methods, or *downcalls*, are functions invoked on the endpoint by some other TME node.
2. Gateway methods, or *upcalls*, are functions invoked by the LCF endpoint to be executed on some other TME node. Because of the endpoint's limited view of the world, these are actually processed on its endpoint gateway

3.2.3.1 Downcalls

The way these operate is very similar to the way method invocations work in the full framework.

Method Invocations in the Full Framework: In the full TME framework environment, any system within the TMR can execute a method on any other system. To do so, it passes the object reference, the method handle and any arguments needed by the method to the object request broker (oserv) on the target node. The target node resolves the object, using the ORB network to locate it, and then invokes the method. The results are passed back to the caller.

Downcall Methods in LCF: The general flow is the same when a managed node invokes a method on an LCF endpoint. However the limitations of the LCF environment place some restrictions on the functions that are possible. Figure 11 shows the process.

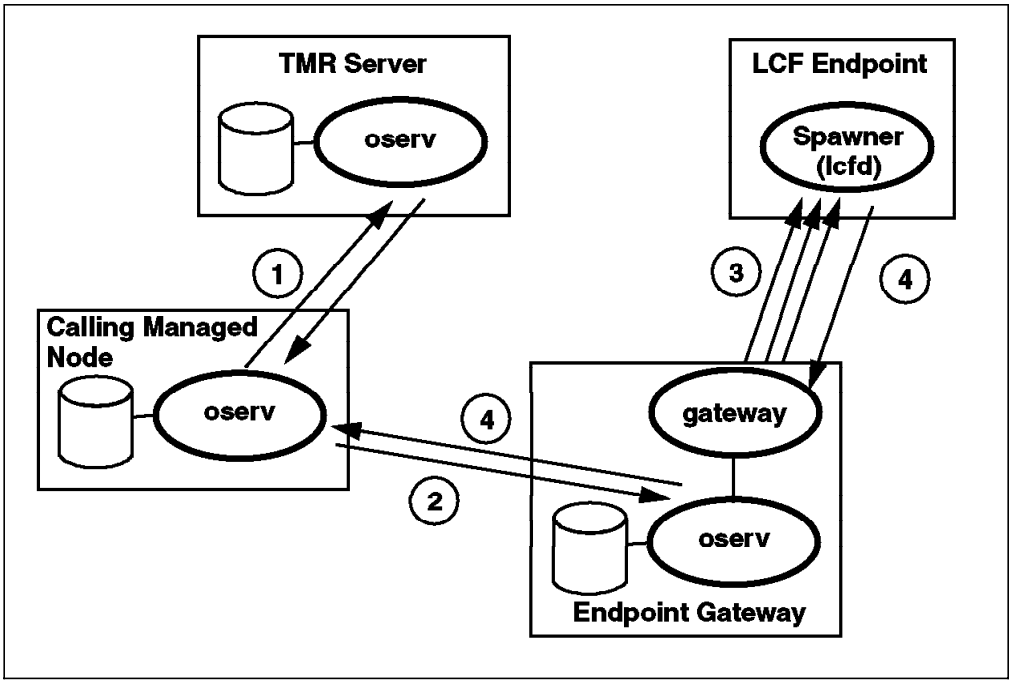


Figure 11. Downcall Invocation

Let us step through the process:

1. The application needs to build a reference for an object on the LCF endpoint. This will involve a call to name resolution services on the TMR server. The

object identifier is in a special format that indicates a downstream resource of the endpoint gateway where the LCF endpoint is logged in.

2. The application passes the object reference and method information to oserv on its host system, which sends it to oserv on the endpoint gateway identified by the object reference.
3. The endpoint does not initially have any method code installed. In this case the gateway first downloads the required files into the spawner's disk cache. The next time the same application is executed, the method code will already be installed, so this step will not be required.

   If the application is updated, the method code stored on the gateway will not match the copy stored in the endpoint cache. The gateway will therefore once again download it before proceeding, overwriting the older version.
4. The method is launched by the spawner. Endpoint methods are simple single-threaded programs. Also, because the endpoint does not have any persistent object store, the objects they work with exist only for the duration of the method. This means that it is up to the application to store any data that it may need for future use.
5. The result of the method invocation are passed back to the gateway, which reroutes them to the calling system in the normal way.

As you can see, LCF places some restrictions on the scope of a method that can be invoked on an endpoint, but the actual invocation process is the same as for the full platform. The application is not aware of the work the gateway is doing to install and maintain the method code.

3.2.3.2 Upcalls

An upcall is the case where an application on an endpoint wants to invoke TME services on another node (managed node, TMR server or another endpoint). The problem, from the point of view of the LCF endpoint, is that it has no way to resolve the object reference in order to be able to send a method invocation. The only contact it has with TME is through its endpoint gateway.

The way this is overcome is by the use of a *mid-level method*. This is a function that is run by the gateway using the services of oserv on the endpoint gateway system. Figure 12 on page 19 shows the flow.

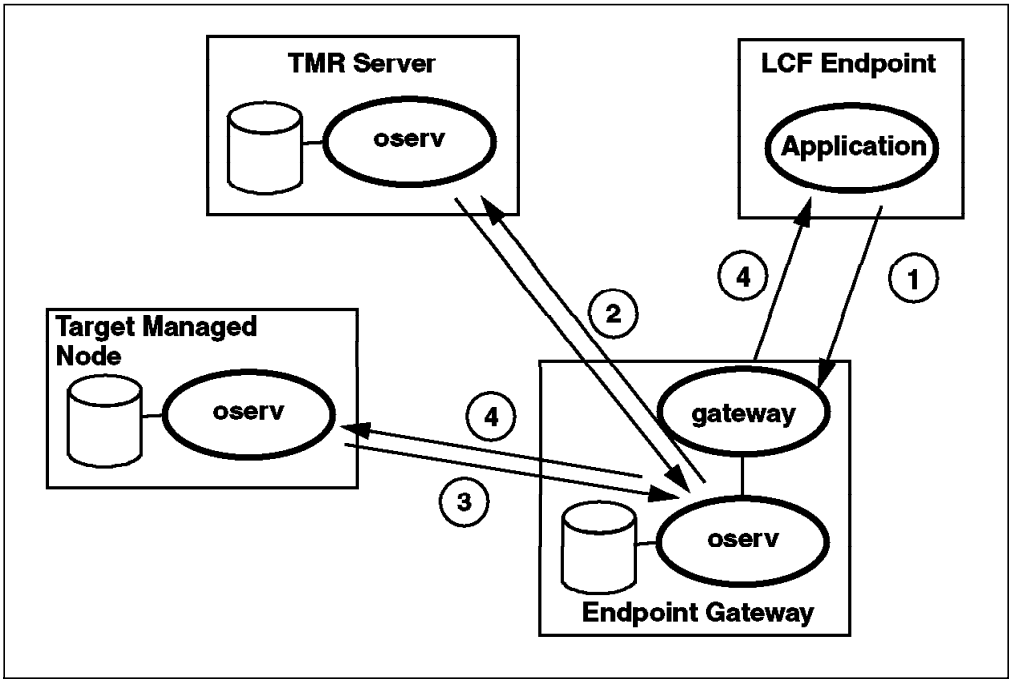


Figure 12. Upcall Invocation

1. The application on the endpoint requests a method invocation on another node. From the application point of view this interface is similar to any other method invocation.
2. The gateway executes the mid-level method. This will need to create an object reference for the target node, so it may need the location services of the TMR server, if the information has not been cached locally on the endpoint.
3. The method invocation is passed to the target system (or endpoint gateway) in the normal way.
4. The response is passed back through the gateway to the LCF endpoint.

Note that, whereas a downcall did not necessarily require the invoking application to be LCF-aware, an upcall will always need additional coding, in the form of the mid-level method installed on the endpoint gateway. Although this is extra work for the application developer it is also another opportunity to improve scalability. The mid-level method can reduce the number of calls to centralized services, by filtering or batching endpoint requests and by caching location data.

3.2.4 How Applications Work with LCF

This discussion of endpoint behavior can become rather abstract. Let us try to make it more concrete by looking at some simple examples.

3.2.5 Downcall Example, Software Distribution

Consider an application that ″pushes″ software to an endpoint node. The file package is constructed on a managed node and sent to the endpoint to be unpacked and installed. In this case the application is only performing a downcall, so no mid-level method is required. The application function will comprise an invoking part, on the source managed node, and a receiving part, comprised of a method to run on the endpoint.

The invoking part will:

1. Use platform services to construct a file package and send it to the endpoint gateway. If it is sending the same package to multiple endpoints on the same gateway, mdist processing will ensure that the data is only sent once.
2. Invoke the receiving method on the endpoint

The receiving method will be installed automatically on the endpoint by the gateway and will then be executed. It will:

1. Retrieve the package from the gateway
2. Unpack it and perform any installation actions, as specified by the arguments passed in the method invocation.

In order to install an application like this you would have to install the invoking part on the calling managed node an the receiving part on the endpoint gateway. The endpoint installation happens dynamically as needed.

3.2.6 Upcall Example, Inventory Scanning

TME 10 Inventory relies on scanning software on the target node to explore the system and generate lists of installed hardware and software. Typically this process happens at boot time, or maybe based on an infrequent timer.

In the LCF environment, a scanner running on the endpoint would have to report its findings by making an upcall. The mid-level method associated with the upcall *could* simply forward the information on to the inventory database application. However, a more efficient approach would be for it to store the data locally and batch together the data for all of its attached endpoints, sending it in one update, rather than in multiple, small updates.

To install an application like this you would have to put the endpoint and mid-level method code on the endpoint gateway. The application would also have to provide an installation process for the scanning application, to place it on the endpoint and automatically invoke it when required.

Chapter 4. Setting Up the Lightweight Client Framework

In this chapter we show an example of setting up a simple LCF environment for Windows 95 and NetWare endpoints. The lab configuration we used is shown in Figure 13

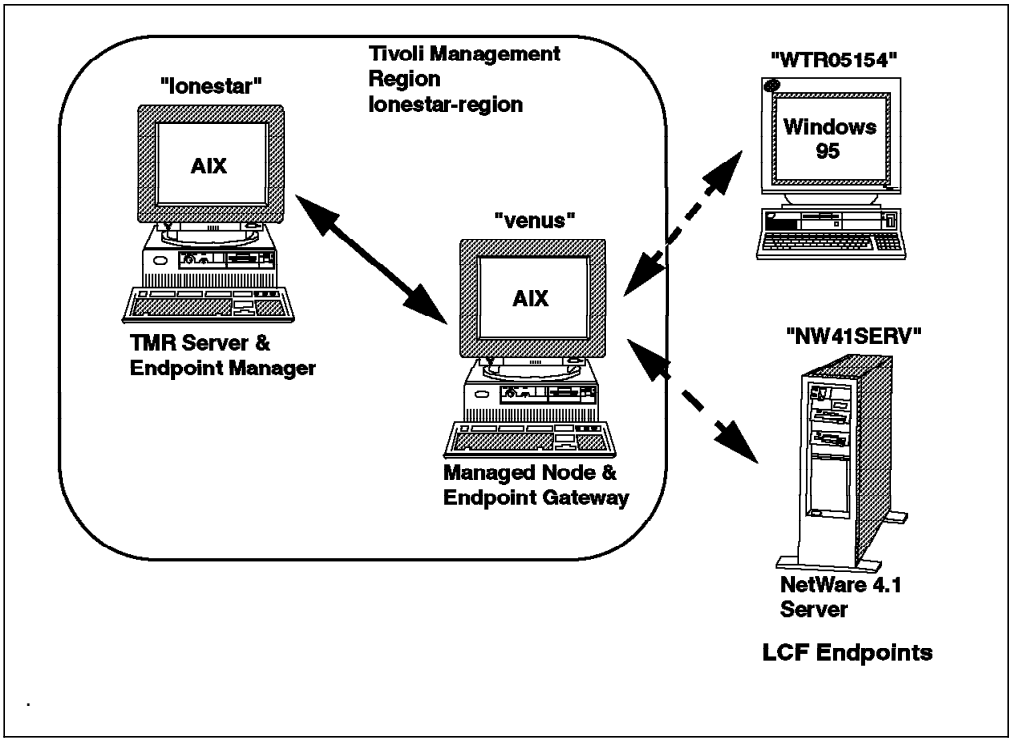


Figure 13. Lab Setup

The two LCF endpoints were both attached to one endpoint gateway, venus.

4.1 Endpoint Manager Prerequisites

Before proceeding with defining the endpoint gateway we installed TME 10 Framework 3.1 on the TMR server and managed node. There is no additional product to provide endpoint support, but you do have to install a patch to the framework, the TME 10 Light Client Framework (LCF) Patch. This patch needs to be installed on the TMR server and on all managed nodes that will be endpoint gateways. After the installation of the patch you will have to restart oserv on all affected systems. The command odadmin reexec all will do this. In our case we found that sometimes after running the command, oserv failed to restart. In those cases simply starting it with odadmin start was effective. If you are running TME 10 Framework 3.2 or later, then the installation of the patch is not required.

Following the installation of the patch you should see a new icon labelled EndPointManager on the root TME 10 Desktop. You can also check for successful installation by using the wlookup command to display the object information for the endpoint manager resource. The command and a typical response is as follows:

```
wlookup EndpointManager
1176566865.1.498#TMF_LCF::EPMgr#
```

If the endpoint manager does not show up, double check that the patch installation was successful. If there is a problem, restore the TME database to a backup prior to the patch install and reinstall it.

4.2 Creating a Gateway

LCF endpoints are self-defining, as we described in Chapter 3, “Introducing the Lightweight Client Framework” on page 13. However, before an endpoint will be accepted by the endpoint manager, you first have to create at least one endpoint gateway that it can connect to. You can create a gateway on any managed node, including the TMR server. We created a gateway that handles NetWare LCF clients as well as Windows 95 clients on venus.

To create a Gateway:

- Right-click on the **EndpointManager** icon. In the pop-up menu, select **Create Gateway** (see Figure 14).

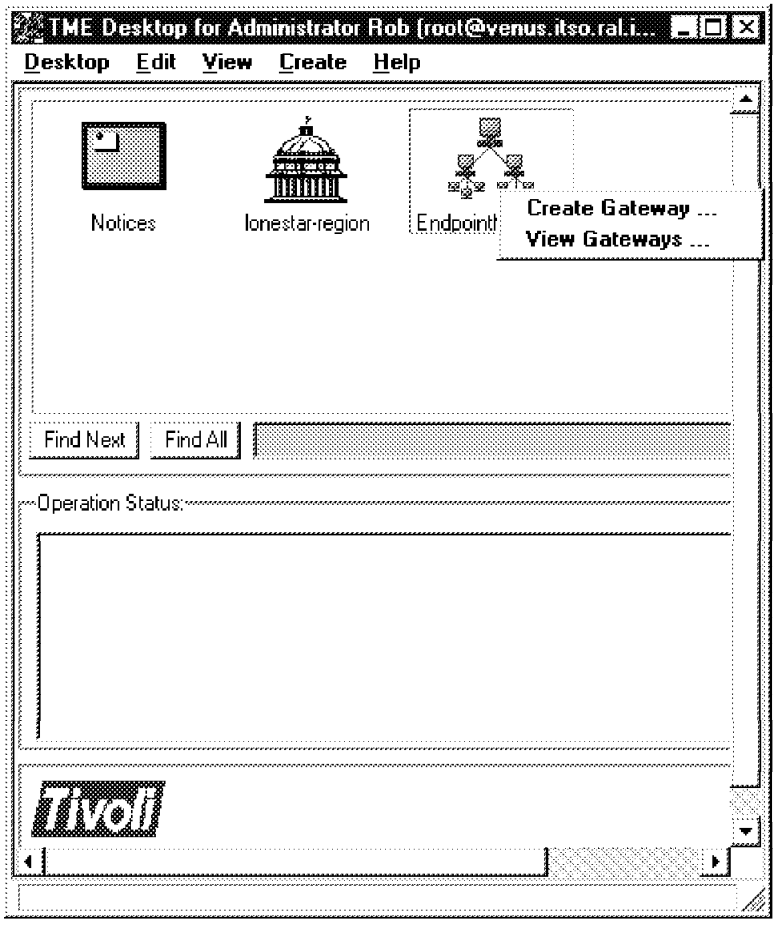


Figure 14. Creating a Gateway

- In the Create Gateway dialog, fill in the required information:

- Provide a name for the gateway. We chose LCF Gateway. If you leave this field blank the default is the name of the gateway node followed by -gateway (venus-gateway in our case).
- The TCP port used for LCF is 9494 by default. Normally you will not need to change this.
- The Managed Node Proxy field specifies the managed node where the gateway will reside. Each managed node can have only one gateway. We chose venus to be our gateway node.

Once you have entered all the data, select **Create & Close**.

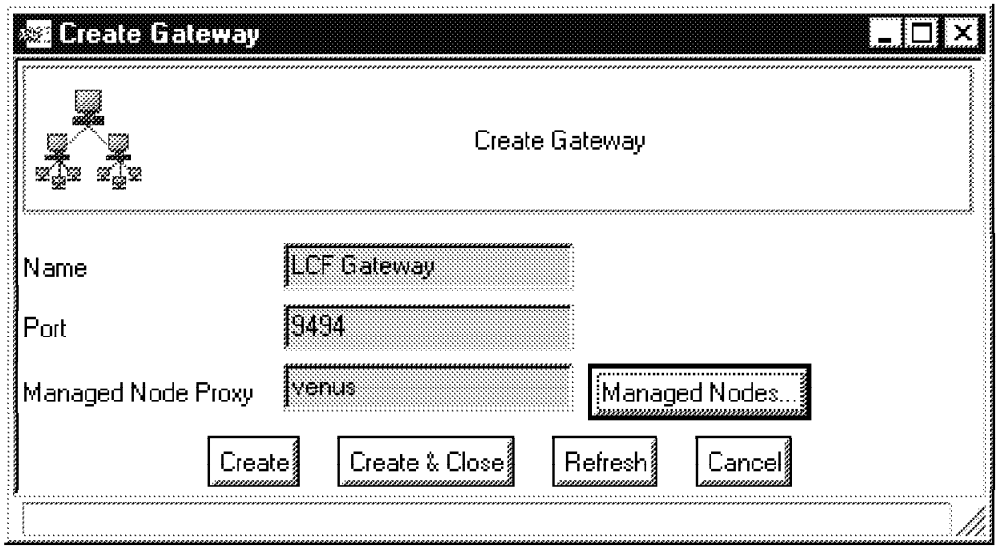


Figure 15. Defining an LCF Gateway

After creating the gateway, double-click on **EndpointManager** on the Tivoli desktop. A gateway definition similar to Figure 16 on page 24 will be displayed.

Creating an Endpoint from the Command Line: You can create an endpoint gateway using the wcrtgate command. The syntax to create the same gateway shown above would be:

```
wcrtgate -h venus -n "LCF Gateway"
```

There are a number of other arguments available for the wcrtgate command:

- You can set the interpreters for endpoints supported by the gateway. For example, nw4 for NetWare Version 4 or win95 for Windows 95.
- You can define the network protocol for the endpoint connection. The only protocol supported in the first release is TCP/IP, so you do not need to specify this.

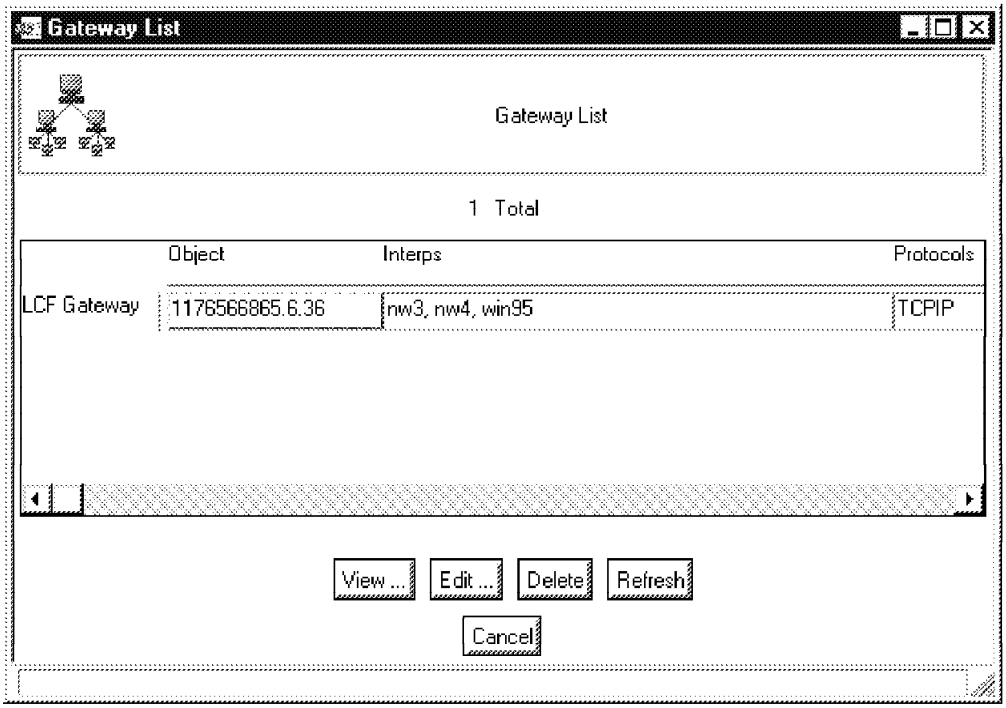


Figure 16. New Endpoint Gateway Appears in Gateway List

Selecting different endpoint gateways for different LCF node types gives you some control over the workload in your network. For more effective control you may need to modify the endpoint gateway policy method, as described in 4.4, “Modifying LCF Behavior Using Endpoint Policy Methods” on page 30

The endpoint gateway function runs as a separate daemon, automatically started by oserv. The daemon is called gateway. You can stop and restart it using the wgateway command.

4.3 Installing and Running LCF Endpoints

Now that we have an endpoint gateway up and running, LCF endpoints can connect to it. We look at two endpoint types, Windows 95 and Novell NetWare Server V4.1.

4.3.1 Installing the LCF Files on Windows 95

The Light Client Framework is installed under Windows 95 just like any other program, using native Windows tools. First you have to make the installation files available to Windows 95. The installation files are on the LCF CD under the win95 directory. If you do not want (or are not able) to load the CD on the endpoint system you can either copy all the files in the directory or access them from a network server. The total size of the install image is about 1.3 MB, so they will fit on one diskette. We used FTP to transfer the installation package to a local directory called lcf. Then we started the setup program by double-clicking on it in Windows Explorer.

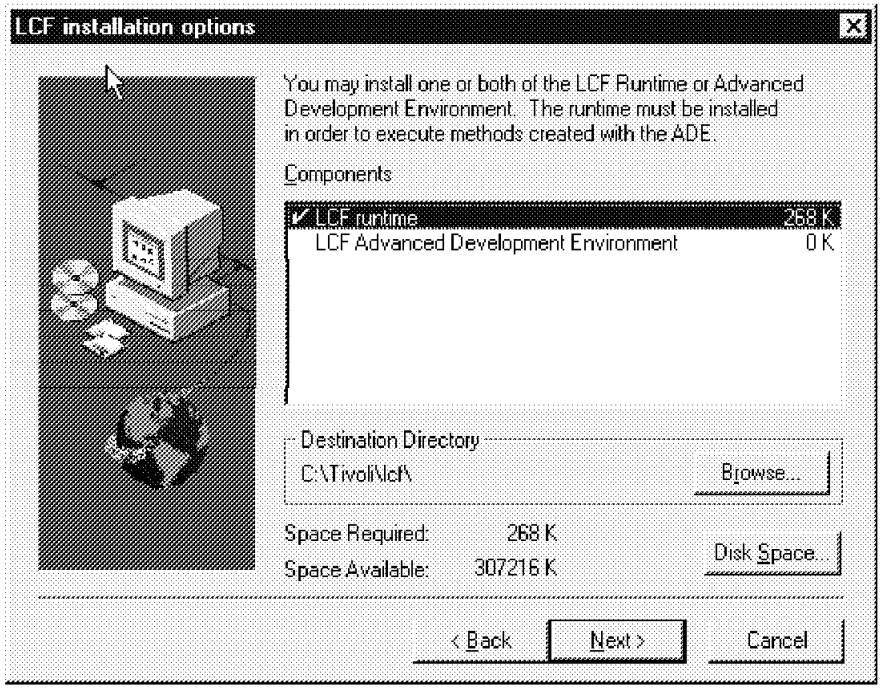


Figure 17. The Initial Install Dialog

The setup program will start the InstallShield Wizard and display the install dialog. If you installed any Tivoli products, such as the TME agent or desktop, before, the install dialog will choose the same subdirectory to install the LCF files. Select **Next** to start the installation process. This will install the LCF files and register the product to Windows 95. You will find that it adds a registry entry under:

```
HKEY_LOCAL_MACHINE->SOFTWARE->Lightweight Client Framework
```

When the install process finishes, reboot your machine. This registers and activates the .PIF definitions required by LCF. LCF tasks will be executed in a Windows command shell using these definitions. After a successful installation, you will see the LCF executables in your Tivoli program group. If you don't have any Tivoli products installed, the install process will create a program group which you can then access by clicking on **Start** and then selecting **Programs** followed by **Tivoli**.

Figure 18 on page 26 shows the Tivoli program group. There are three icons to control the launch the LCF daemon in this pre-release version:

lcfd The normal lcf executable. It runs in background and writes into a logfile called lcfd.log.

lcfd-debug A debugging version of lcfd. It opens a console window where you can watch the logging messages issued by lcfd. The default debug level for this selection is level 2, but you can change it to a higher value if you want by editing the properties for the shortcut and changing the argument on the -d switch

stop lcfd This shortcut stops lcfd and closes the window it is running in.

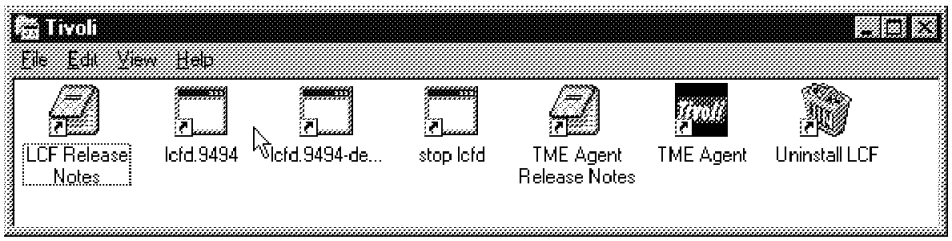


Figure 18. Tivoli Program Group after LCF Install

Note

The lcfd.9494_debug icon just starts the console, which seems to be equivalent to debug level 2 when running LCF under NetWare. If things don't work as expected, you can increase the debug level. Right-click the lcf_debug icon and select Properties. In the shortcut tab, you will find the target entry. Edit the command line and add a -d 9 between the program path and the -s flag.

Normally you would want the LCF daemon to be restarted whenever the system is booted. You do this by putting a Windows 95 shortcut to the daemon startup icon in the **Startup** folder. From the Windows desktop, select **Start** with the right mouse button and then select **Open** from the menu**.** Double-click on **Programs** and then **Startup** to open the Startup folder. You can now create a shortcut to the LCF daemon by selecting **File - Create - Shortcut** from the menu bar and using the file browser to locate the lcfd executable. By default it is Tivolilcfbinwin95mrtlcfd.exe.

4.3.2 Running LCF on Windows 95

After you successfully installed the LCF files, you can launch the LCF daemon by selecting it from the Start menu. For the first startup, you might start LCF in debug mode to verify its correct operation. Click on **Start** and then select **Programs** followed by **Tivoli** and then **lcfd_debug**. The LCF console will start and will display a number of messages similar to Figure 19 on page 27.

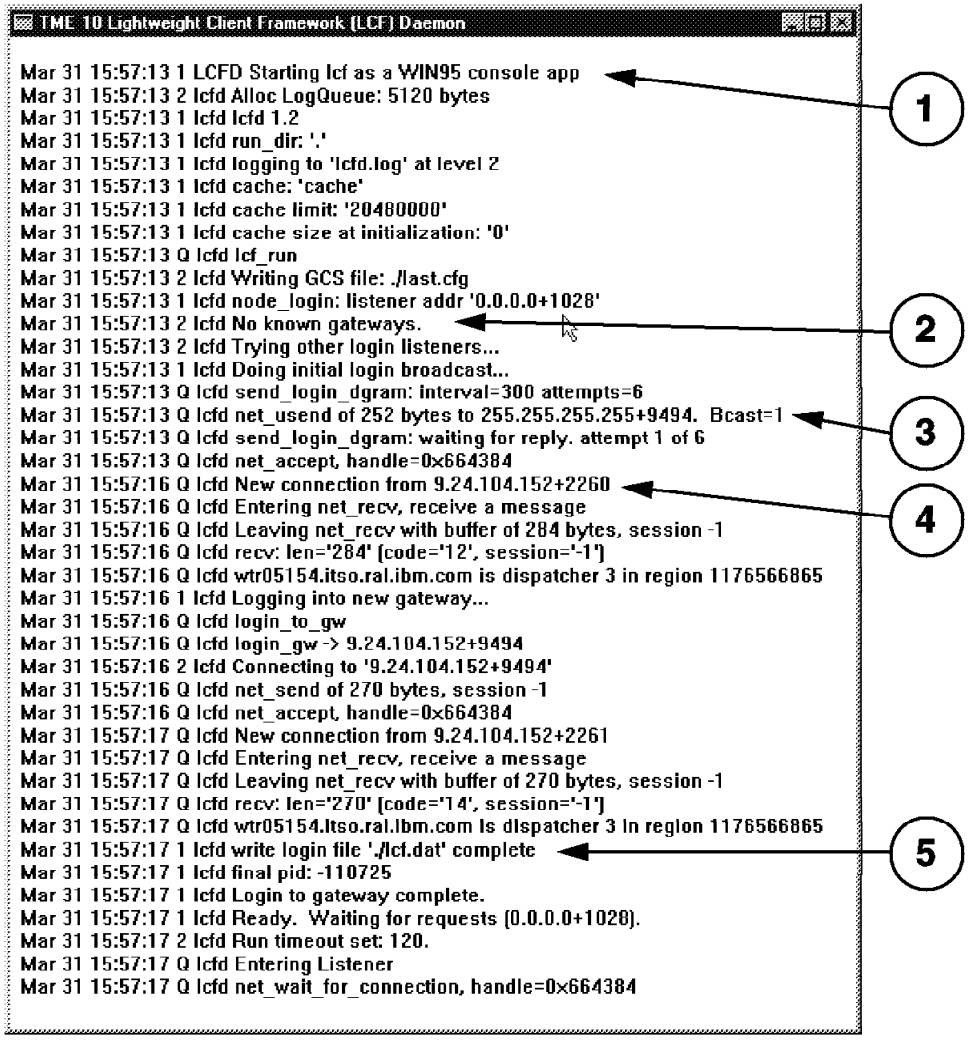


Figure 19. The LCF Console

1. The LCF daemon starts in a foreground window. The first few messages are all related to normal program startup
2. LCFD looks for an existing configuration file, lcfd.dat. In this case it does not have an existing configuration and therefore does not have an assigned gateway.
3. LCFD broadcasts a gateway request to port tcp/9494, the default gateway port. Any gateway listening will receive this broadcast, but it will not respond to it. Instead it will forward the gateway request to the endpoint manager (on their local TMR server) which will choose a gateway based on gateway configuration and local policy.
4. LCFD receives a response from the gateway that was selected by the endpoint manager. It then goes on to establish a connection to the gateway, registering endpoint details and exchanging encryption keys.
5. Now that a gateway is assigned, LCFD will continue to attempt to use it whenever it is restarted. It stores the gateway details into a configuration file, lcf.dat.

You can also see the effect of a successful connection to a gateway from the TME desktop. From the desktop click on the **EndpointManager** icon with the right mouse button, select **View Gateways ...** and select the gateway entry which serves as a gateway for Windows 95 nodes. Choose the **View ...** button. This will bring up the list of all endpoints assigned to that gateway where your machine should appear (see Figure 20).

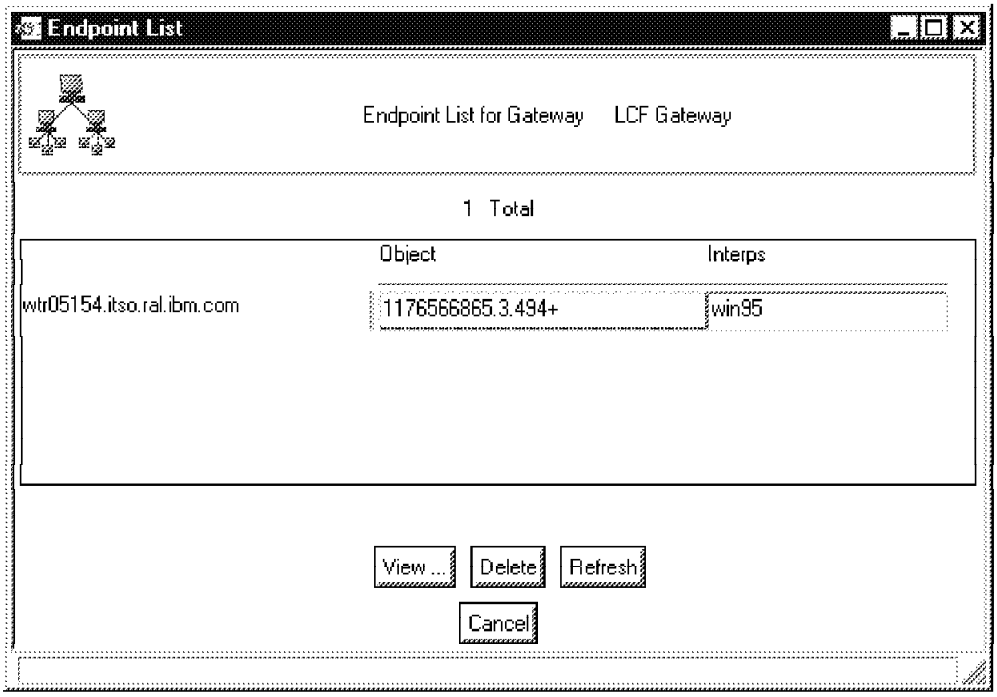


Figure 20. New LCF Node Appears in the Endpoint List

4.3.3 Installing LCF on NetWare

The installation process for the NetWare LCF daemon must be performed from a NetWare client running under Windows. We used the Novell Windows 95 client software, rather than the built-in NetWare support from Microsoft, although any client in a 32-bit Windows environment should be acceptable.

The installation files for NetWare are found on the LCF CD, under the nw3 and nw4 directories. We used a NetWare Release 4.1 server in our example. In order to install you first have to map the NetWare server root directory (the SYS: volume) to a drive under Windows and access it using supervisor authority. We used the *Supervisor* user ID, but it should also be possible to use another ID with appropriate NetWare authorization.

To install LCF, select the install directory in Windows Explorer and double-click on the **Setup** program. This will then invoke the familiar InstallShield Wizard. One piece of information required by setup is the directory path for the LCF installation. The default for this field is C:TIVOLILCF, but you need to specify a subdirectory under the mapped SYS: volume on the NetWare server. In our case we chose to create a new directory named TIVOLI, as shown in Figure 21 on page 29. To complete the installation, click on **Next**.

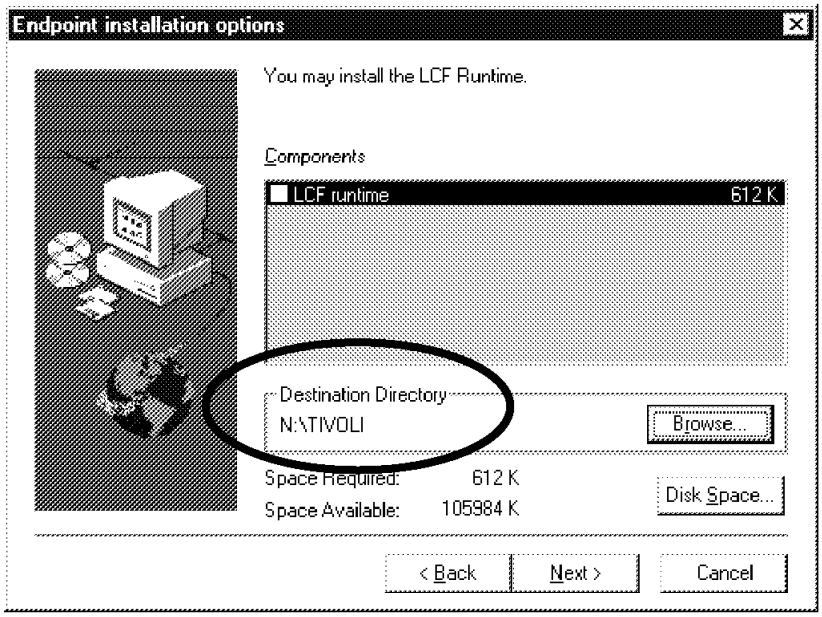


Figure 21. LCF Install for NetWare, Showing Destination Directory

The install process creates a directory hierarchy under your chosen directory, as shown in Figure 22.

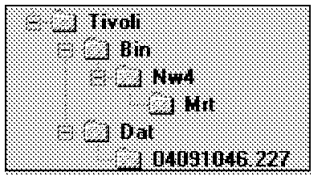


Figure 22. NetWare Server Directory Tree after LCF Installation

4.3.4 Running LCF on NetWare

You will find the LCF NetWare Loadable Module (NLM) in the BINNW4MRT directory below the destination directory that you specified above. Normally you would add the command to load the NLM to the AUTOEXEC.NCF file, so it will be loaded every time the server restarts. For the first time, you might want to start it manually. To do so, enter the LOAD command with the full NLM path at the local console or from a remote console with *Supervisor* access to the NetWare server.

In our example, the full command is:

```
LOAD SYS:TIVOLIBINNW4MRTLCFD.NLM
```

When the NLM starts it displays a formatted panel containing the same sequence of messages seen in Figure 19 on page 27. Figure 23 on page 30 shows the NLM console.

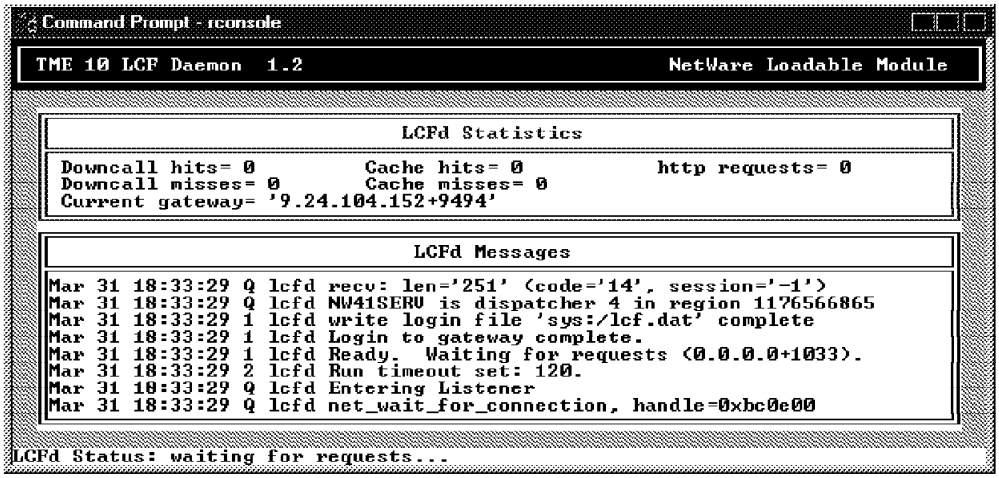


Figure 23. The LCF Client Console

After a correct startup, your console window shows the LCF messages last issued by lcfd. In addition, the upper part of the console window displays the gateway to which your NetWare client is connected. If, for any reason, you need to examine all messages, you can just open the logfile. By default, LCF writes the logs into the root directory (SYS:). Now two endpoints appear below the endpoint gateway icon on the Tivoli desktop (see Figure 24).

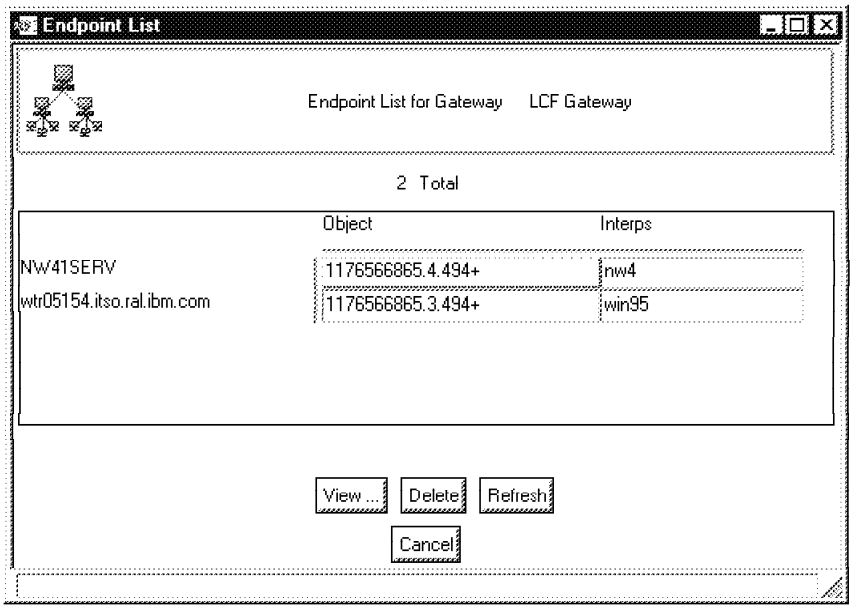


Figure 24. The NetWare Endpoint Appears

4.4 Modifying LCF Behavior Using Endpoint Policy Methods

Endpoint policy methods can be used to take actions at certain points when an LCF endpoint is interacting with an endpoint gateway. The policy methods are similar to standard TMF policy methods, however, there is a special set of commands to deal with the policy methods for endpoint policies.

While for the standard TME framework policy methods you will use commands, such as wgetpolm and wputpolm, the commands wgeteppol and wputeppol are used for endpoint policies.

The following endpoint policies are available:

- allow_install_policy
- after_install_policy
- login_policy
- select_gateway_policy

In terms of the TME framework, allow_install_policy is a validation policy, whereas the others are default policies. The validation policy allow_install_policy can be used to determine if an endpoint that wants to be defined at an endpoint gateway is going to be accepted.

As with standard policies, the endpoint policies can be implemented as any form of executable which will most likely be a shell script. In case of allow_install_policy the script returns 0 if the endpoint is permitted to be registered with the desired gateway, or 1 if not. This can be used, for example, to deny machines of a certain type, a certain name, or from a certain IP subnet to be registered with an endpoint gateway. Or, you might want to maintain a list of all the possible endpoints that are allowed to connect.

There could be good reasons for implementing your own allow_install_policy method. As LCF endpoints are ″self-defining″ there is no strict security control in a way that the TMR server automatically restricts which endpoints can be registered.

Think of the following scenario: you might want to modify the after_install_policy in a way that a new endpoint is automatically subscribed to a certain profile manager, let's say one that is used for distributing software packages to Windows NT machines. This profile manager might be itself subscribed to another profile manager for grouping purposes. If you could register your LCF endpoint to the gateway and therefore automatically subscribe it to the profile manager (assuming the policy to do that is in place) you could possibly gain access to a file package that you are not supposed to have access to.

The following figure gives an overview of the endpoint policies:

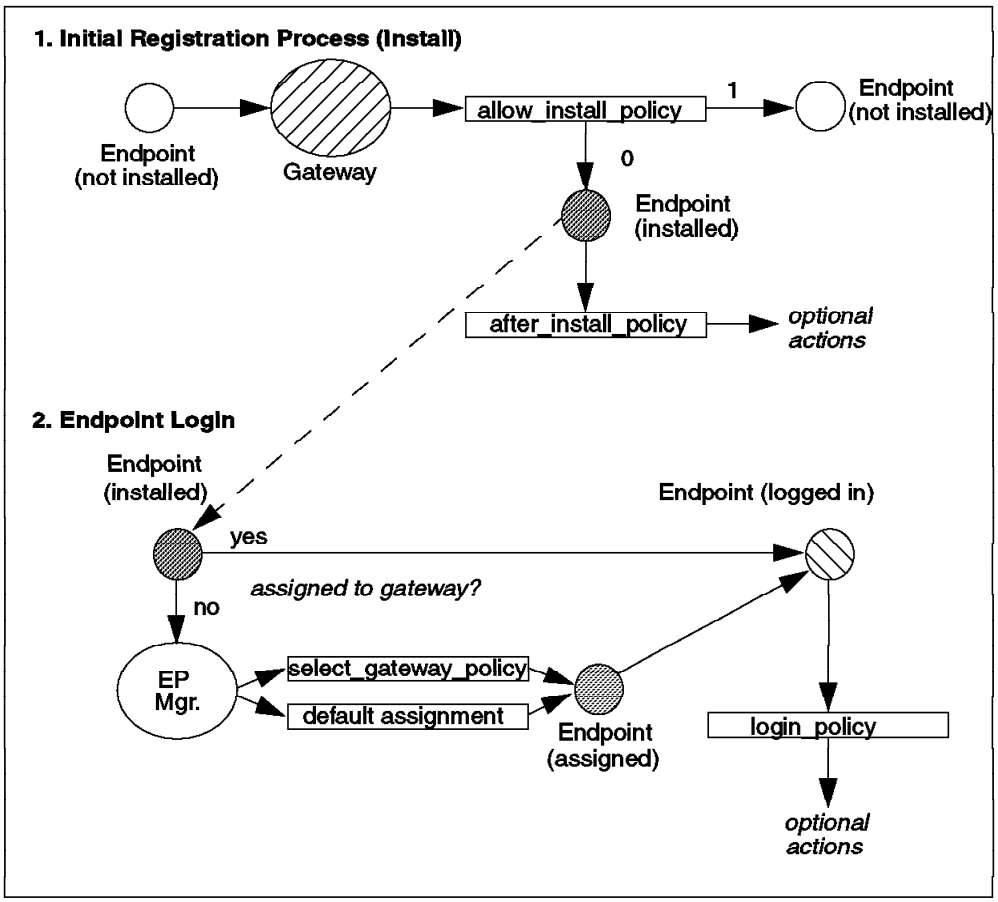


Figure 25. LCF Endpoint Policies

The policies allow_install_policy and after_install_policy are called only during the initial registration of the endpoint with a gateway. If there is an allow_install_policy and the endpoint is rejected by that policy, then the registration will be stopped. Otherwise, the endpoint will be registered (installed) and optional actions can take place, triggered by the after_install policy. The most common action here might be to automatically subscribe the new endpoint to an existing profile manager.

When an endpoint initially needs a gateway assigned or has lost its assigned gateway, the endpoint manager has to assign an endpoint gateway. If no select_gateway_policy exists, the endpoint manager uses its default mechanism to determine the gateway, otherwise the policy script in the select_gateway_policy has to determine the gateway. If this script fails, the default mechanism will be tried. The policy can either return a single gateway or a list of gateways to try.

For example, the select_gateway_policy could be implemented in a way that all Windows NT machines are routed to one gateway and all AIX machines to another. Or, you might want to use the IP subnet in which the endpoint and gateway reside as a criteria. In that case you might also want to combine the gateway selection with collections used in TME 10 NetView, for example, by using the select_gateway_policy to ensure that nodes in the same subnet are also group under the same mid-level manager.

Finally, the login_policy is called, whenever an endpoint connects (logs in) to a gateway. This can be used, for example, for logging all connections to a file, or the Tivoli notice board.

We show a very simple example for the allow_install_policy: whenever a Windows NT LCF client tries to get installed we want to reject it and send a message to the TME Administration notice board.

To retrieve the current policy script for the allow_install_policy we type:

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

This will store the existing policy script in the file allow.sh. The body of the script is empty by default. We replace allow.sh with the following content:

```
#!/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.
#
if [ "$3" = "w32-ix86" ]
then
  echo "**** WARNING ****" >/tmp/notice
  echo "The following node has tried to connect to gateway $4  ">>/tmp/notice
  echo "and was rejected:" >>/tmp/notice
  echo "$1 (object id=$2, ip address=$5)" >>/tmp/notice
  echo "**** WARNING ****" >>/tmp/notice
  wsndnotif "TME Administration" Notice </tmp/notice
  rm /tmp/notice
  exit 1
fi
exit 0
```

When the client tries to get installed, the allow_install_policy script is invoked and passed certain parameters describing the client. One of these parameters is the architecture type of the client, which is w32-ix86 in the case of a Windows NT client. If the architecture type is w32-ix86 we create a temporary file /tmp/notice that contains a warning message and then send a notice to the TME notice group TME Administration. Finally, we exit the script with return code 1, indicating that the client is rejected.

To activate the new method, we type:

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

To test the script, we install the LCF client code on a Windows NT 4.0 system, victor in our case. After the installation has finished, we start the lcfd daemon on the client. The client tries to connect to our existing gateway, but is rejected.

In the TME Administration notice group we receive the following message:

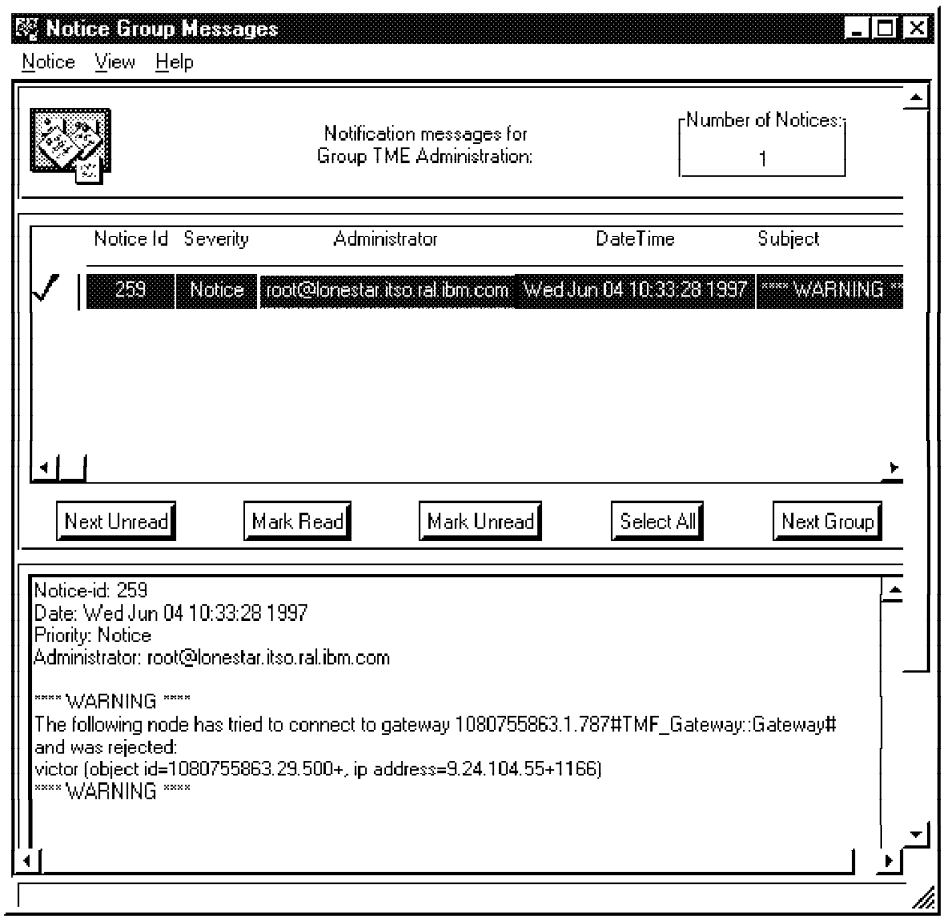


Figure 26. Notice Group Message When LCF Client Is Rejected

As a further example, we want to automatically subscribe a new LCF client to a profile manager. We want to subscribe all new Windows NT clients to a profile manager called WinNT. Therefore, we first have to remove the allow_install_policy again, that does not permit Windows NT clients. You can do this, by just replacing allow.sh with an empty script again and typing:

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

Before we modify the after_install_policy we create the new profile manager:

```
wcrtprfmgr lonestar-region WinNT
wsetpm -d /Library/ProfileManager/WinNT
```

The first command creates the profile manager, the second enables it to handle dataless endpoints, that is LCF clients.

To perform the automatic subscription we create the following script in after_install.sh:

```
#!/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.
#
if [ "$3" = "w32-ix86" ]
then
  wsub /Library/ProfileManager/WinNT @Endpoint:$1
fi
exit 0
```

The script checks again for the architecture type w32-ix86 and in case this condition is met, automatically subscribes the new Windows NT LCF client to the profile manager WinNT.

Shortly after starting the lcfd daemon on the Windows NT machine victor again, the node will be subscribed to the profile manager:

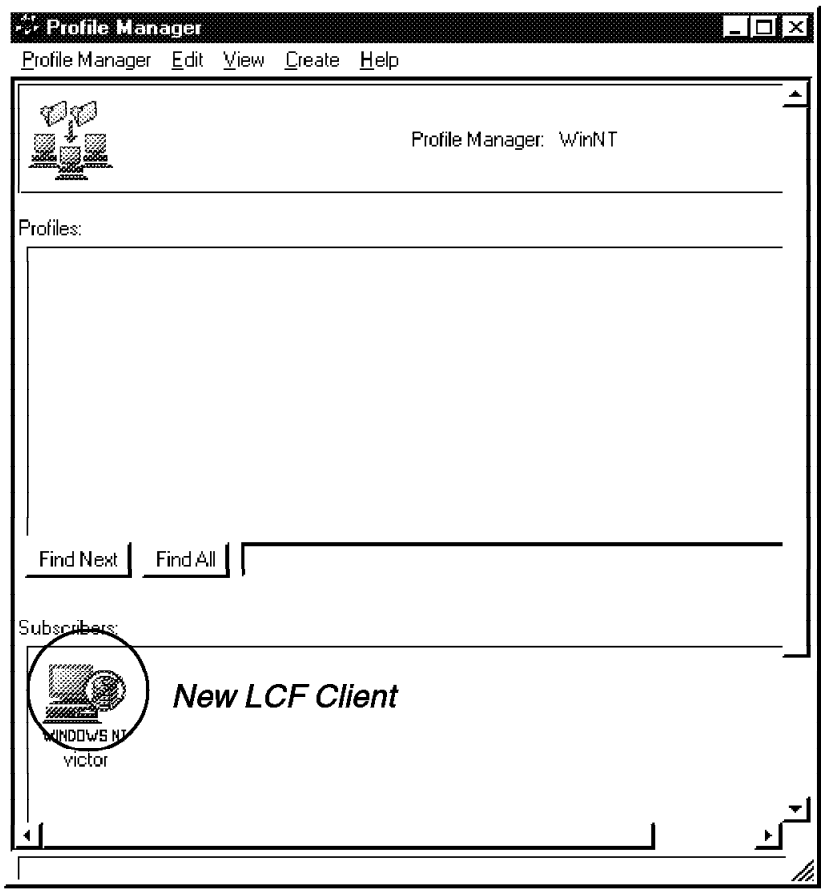


Figure 27. Automatic Subscription of LCF Client

4.5 Execute Tasks on LCF Endpoints

Note

At the time this redbook was written, it seems a Windows 95 task must adhere to the following rules:

The task definition must consist of a single command to be executed. Otherwise a ″too many parameters″ error will occur.

On Windows 95, the task gets executed in a win32s console, the .PIF description file for which is provided by the LCF installation process. That limits the useful programs to those that do not require any manual intervention and, in addition only use Standard I/O streams to do input and output. Launching a Windows 95 executable via LCF would start that executable but the task would never return.

To verify our LCF setup, we executed a task on the Win95 target node.

The following steps assume there is a policy region present and the gateway node is part of that policy region . In our case, we created a new policy region called Light Stuff. We set the managed resources of the new region to include task libraries. Then we created a task library and task, as follows:

1. Create a task library in the policy region by selecting **Create** followed by **Task Library** from the menu bar in the policy region window. We called our task library Really Simple Tasks.
2. We want to execute a simple DIR command on the Windows 95 node. Before we can do this, we needed to place the command into a command script which must reside on a managed node.
   To create the command script, use your favorite text editor and produce a file containing the command to be executed. We entered the following command to create the file on UNIX:

   ```
   echo ″dir c:*.*″>/tmp/lcf_w95_test1
   ```

3. Then double-click the task library icon and select **Create Task** from the menu bar. In the create task dialog, choose Windows 95 as the platform. The executable for that task is the command file you just created. Provide the complete path and click **Create & Close** to create the task. Don’t forget to give the task a name. Figure 28 on page 37 shows the definition we used.

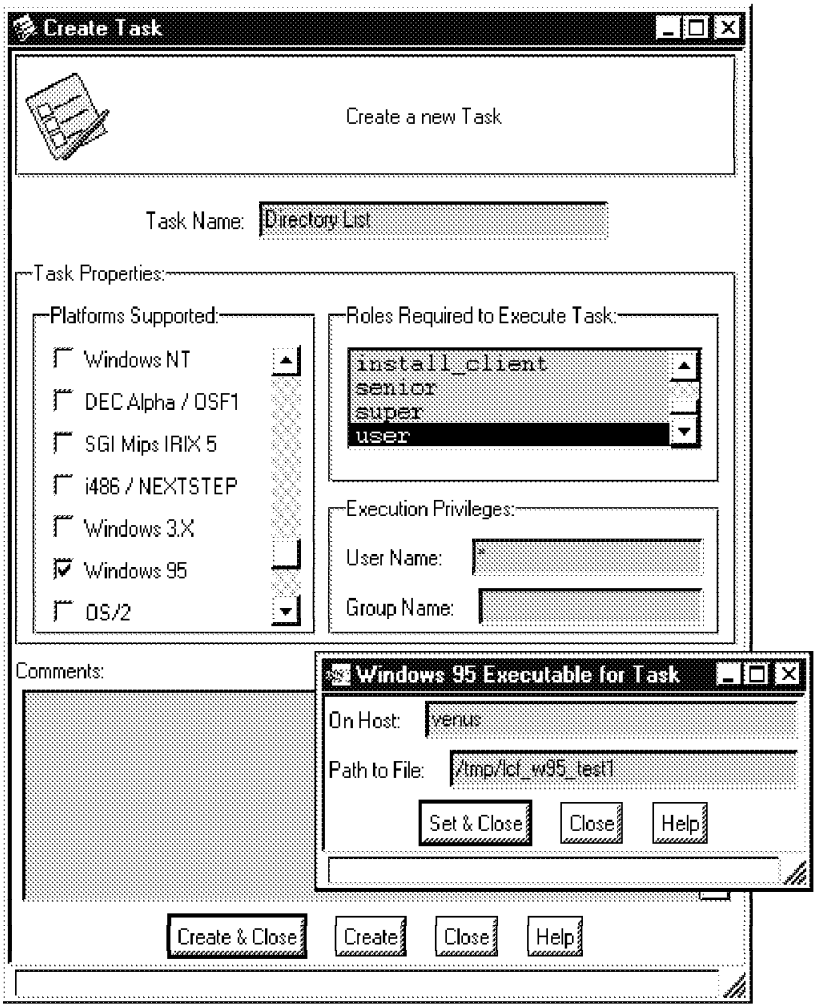


Figure 28. Defining a Task

4. Finally you can execute the task on a selected endpoint. Select the new task icon with the right mouse button and select **Execute** from the menu. The LCF endpoints will appear in the Available Task Endpoints list, so you can select the systems you want to execute the task on (see Figure 29 on page 38).

.

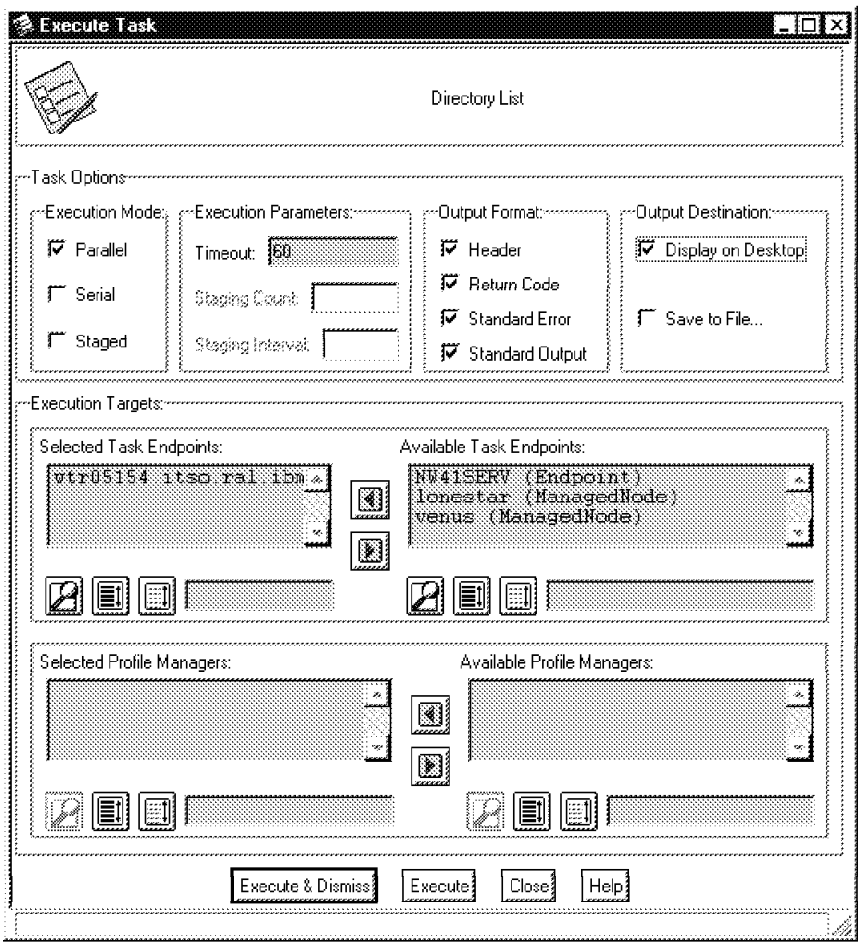


Figure 29. Executing a Task on a Windows 95 LCF Endpoint

Click on **Execute and Dismiss** to invoke the task on the managed node. The result of the DIR command appears in the execution window (or in a file, if you did not choose the Display on Desktop option). See Figure 30 on page 39.

.

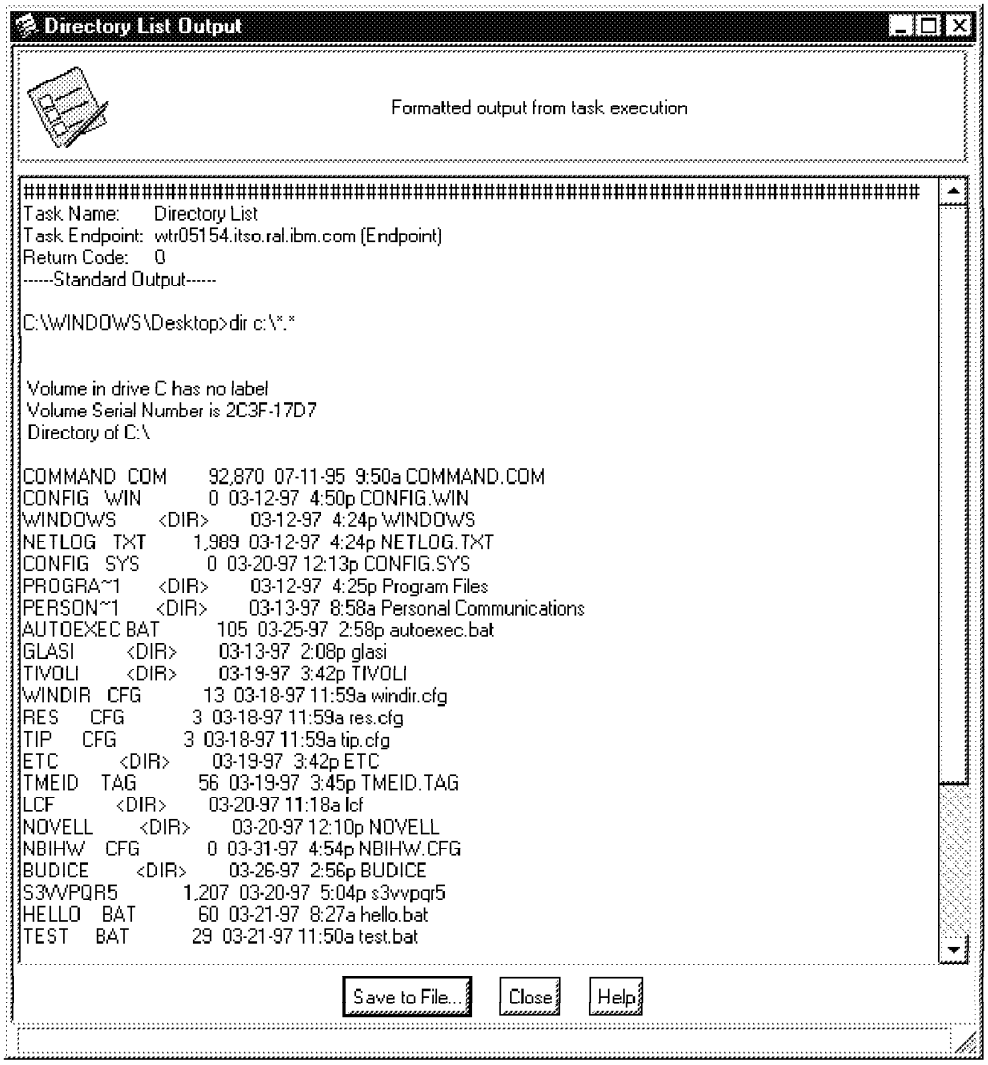


Figure 30. Output of the Windows 95 DIR Command

Chapter 5. Sentry Meets the LCF Endpoint

From Version 3.5 of TME 10 Distributed Monitoring, LCF endpoint support will be included. However, in the initial release the only supported type of endpoint will be NetWare servers.

From a TME 10 point of view, NetWare is a special kind of platform. Originally designed to provide disk sharing for PCs in a LAN, NetWare lacks some of the features you expect to find in a normal operating system. In particular, NetWare has very limited command line interface support for running programs or scripts. The programming model of netware is based on NetWare Loadable Modules (NLMs). These are modules which can be loaded dynamically but are intended to load and run without exit.

NetWare endpoint support is therefore implemented as a number of NLMs, which are spawned by the LCFD daemon (also an NLM). A number of system monitors are provided, allowing you to examine different aspects of the performance and status of a Novell server. Table 1 lists the monitors that are available.

GUI Name	CLI Name	Description
CPU Utilization	CPUUtilization	Percentage CPU load on server
Process Count	ProcessCount	Number of processes currently running on the server
Thread Count	ThreadCount	Number of threads currently in use by running processes
Volume Space Used	VolumeSpaceUsed	Size, in megabytes of a given disk volume (SYS: for example)
Volume Space Remaining	VolumeSpaceRemaining	Free space in megabytes on a given disk volume
Cache Blocks in Use	CacheBlocksinUse	Amount of memory given over to disk caching, in number of blocks
Percent of Cache in Use	PercentofCacheinUse	Percentage of cache space currently in use
Number of Connected Users	Numberof connectedUsers	How many users are currently using the server

Table 1. List of NetWare Monitors

Note that the GUI Name and CLI Name columns refer to the TME managed node from which you create and control the monitors. TME 10 Distributed Monitoring provides no graphical or command line interface on the monitored NetWare server itself.

5.1.1 Installation Notes

TME 10 Distributed Monitoring 3.5 requires the Lightweight Client Framework on the NetWare server. Installation uses the standard TME product installation process. As with previous versions of Sentry, the base application and the monitoring collections are shipped as separate installation packages, so you only need to install the monitors for the environments you need.

You should note, however, that the NetWare monitoring collection is unlike other monitoring collections, in that it needs to be installed on the TMR server *and* the endpoint gateway(s). Usually monitoring collections only have install components for the server, but in this case there are endpoint methods that have to be present on the gateway to allow the spawner to access them. As the other

monitoring collections are modifed to support LCF, we expect to see this requirement become common across the board.

5.2 Creating an Endpoint Enabled Profile Manager

NetWare endpoints cannot be controlled by normal profile managers. They need a special instance of a profile manager which supports the *dataless endpoint mode* (in other words, LCF).

To create an endpoint-capable profile manager, follow these steps.

1. In your selected policy region, select **Create ProfileManager.**
2. Give the profile manager a name.
3. Select the **Dataless Endpoint Mode** check box and click on **Create & Close**. Figure 31 shows the definition that we used.

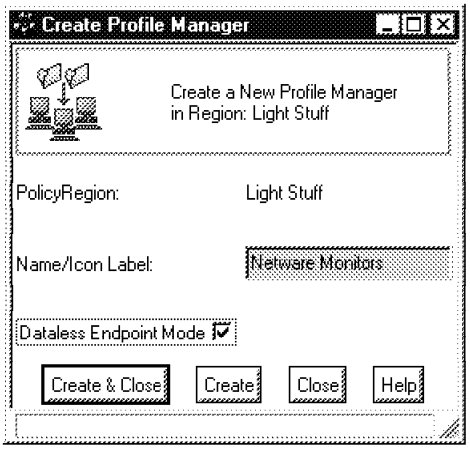


Figure 31. Defining a Profile for an LCF Endpoint

The new profile will show up in your policy region window. It should have the LCF Endpoint icon, as shown in Figure 32. If it differs, you probably did not select the Dataless Endpoint Mode upon creation.

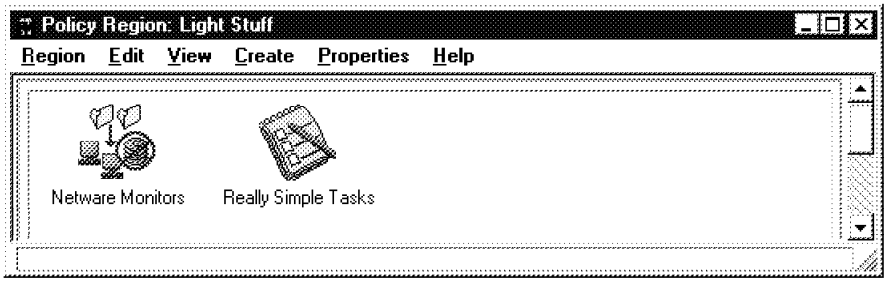


Figure 32. The New Profile Manager

Once you have created the endpoint enabled profile manager, you can start to configure and define the profiles for your endpoints.

5.3 Sentry and the NetWare Endpoint

If you are familiar with defining Sentry monitors, working with TME 10 Distributed Monitoring on LCF endpoints is very much like working with normal managed nodes. However, there are some differences.

- Because endpoint nodes are different from normal managed nodes, only limited support is given. In addition, the tools normally available might not be present on the target node.
- The NetWare server operating system has limited ability for executing command line programs. NetWare executables are in the form of NetWare Loadable Modules (NLMs). You can load and unload them dynamically but, once loaded, the modules are part of the operating system. This means that you cannot define a program as an automated response to a monitor.
- You do not have to install any additional code on the system. However, you do need the LCFD NLM to be running on the endpoint, and an endpoint relationship must exist between the NetWare server and a Tivoli Endpoint Gateway. If this relationship has been established successfully you can start distributing monitors to NetWare.

We now step through the process of defining a sample distributed monitor for NetWare.

5.3.1 Creating a Monitoring Profile and Setting Subscribers

We place our new profile in the profile manager we created in 5.2, “Creating an Endpoint Enabled Profile Manager” on page 42.

1. Double-click the profile manager icon and in the profile manager window; select **Create** and then **Profile** from the menu bar. Select **SentryProfile** from the list of available profile types, give the profile a name and click on **Create & Close**. If you do not see SentryProfile listed it is probably because you have not added the class of profile as a managed resource of the policy region. Return to the policy region and select **Properties** followed by **Managed Resources** from the menu bar and add the TME 10 Distributed Monitoring resources.
2. First, define the nodes that will subscribe to the new profile. Select **Profile Manager** and then **Subscribers** from the profile manager window menu bar. If you correctly set up your endpoint gateway and the NetWare server is registered to that gateway, you will see it as an endpoint in the list of possible subscribers (see Figure 33 on page 44).

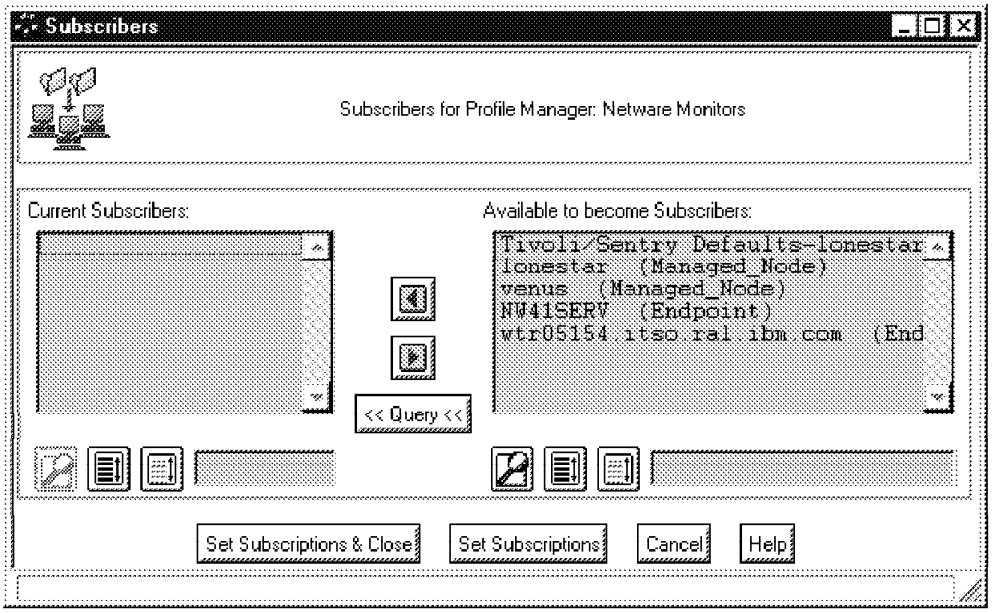


Figure 33. Subscribing an LCF Endpoint

3. Select the NetWare Server and click on **Subscribe & Close**. As a result, your profile manager now shows the endpoint in its subscribers field, as shown in Figure 34. Note the new endpoint icon.

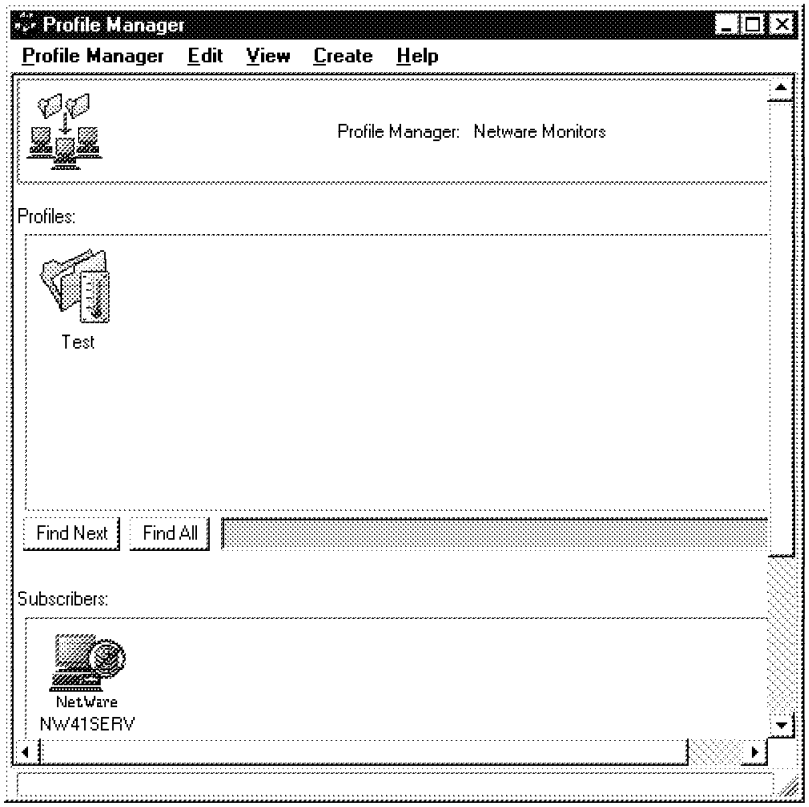


Figure 34. The New Profile and Its Subscriber

From now on, working with the endpoint node is almost identical to the normal tasks you would do when defining any other Sentry monitor:

1. Choose a monitor out of the appropriate group.
2. Provide optional parameters.
3. Set properties such as response level, type of notice and monitoring schedule.
4. Save the new definition.
5. Set the distribution defaults.
6. Distribute the monitor.

The following section steps through this process for a NetWare monitor.

5.3.2 Defining a Monitor for a NetWare Endpoint

We now define a sample NetWare monitor and distribute it to the NetWare server. We assume you already know how to work with Sentry monitors and will concentrate on the differences.

1. In your profile manager window, double-click the icon for the profile you created earlier. In the list of monitoring collections look for NetWare Monitors (see Figure 35). If the collection does not show up, it may be that you did not install the NetWare Monitors component when you performed the TME 10 Distributed Monitoring installation. See 5.1.1, “Installation Notes” on page 41.

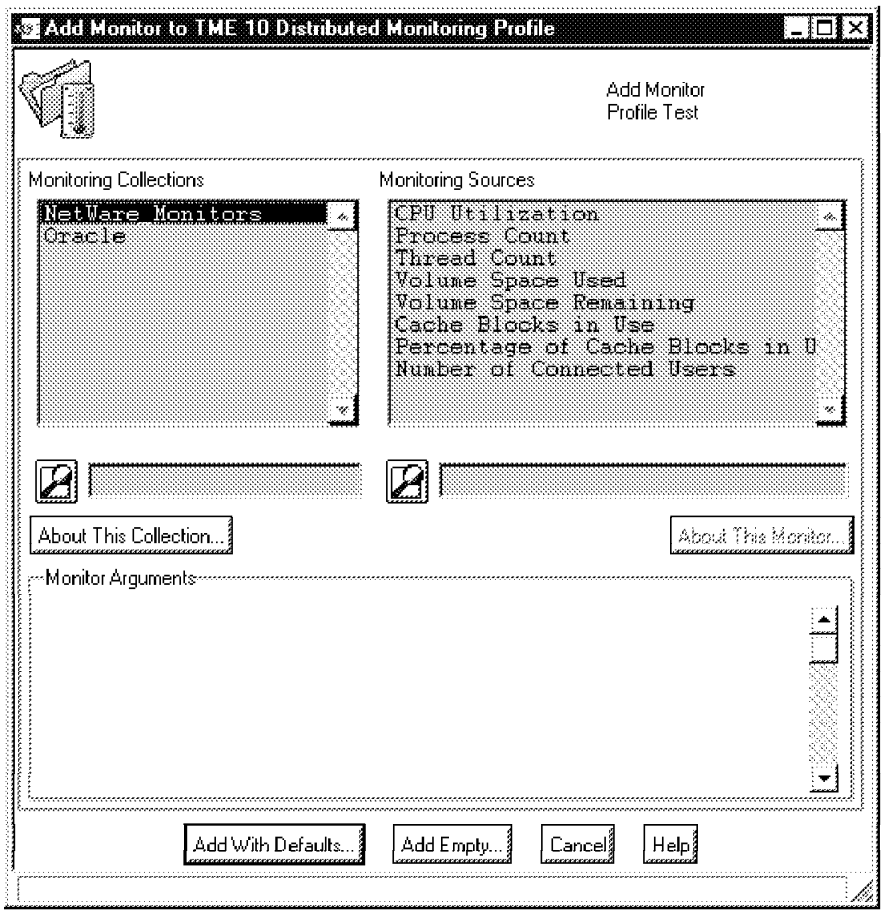


Figure 35. The NetWare Monitoring Collection

Some of the provided monitors for NetWare endpoints require a parameter to be specified. Keep in mind that the NetWare operating system uses its own naming conventions for volumes and pathnames.

2. For our sample monitor, we selected **Volume Space Remaining**, defined a path name of SYS: and then clicked on **Add Empty...**, which finishes the monitor selection and leads you to the Edit Monitor dialog.
3. We configured the monitor to send a severe condition when the free space in the volume drops below 95 MB. We set actions to raise a Sentry notice, pop-up a message on the administrator's desktop and also set an indicator icon. Figure 36 shows the monitor definition screen

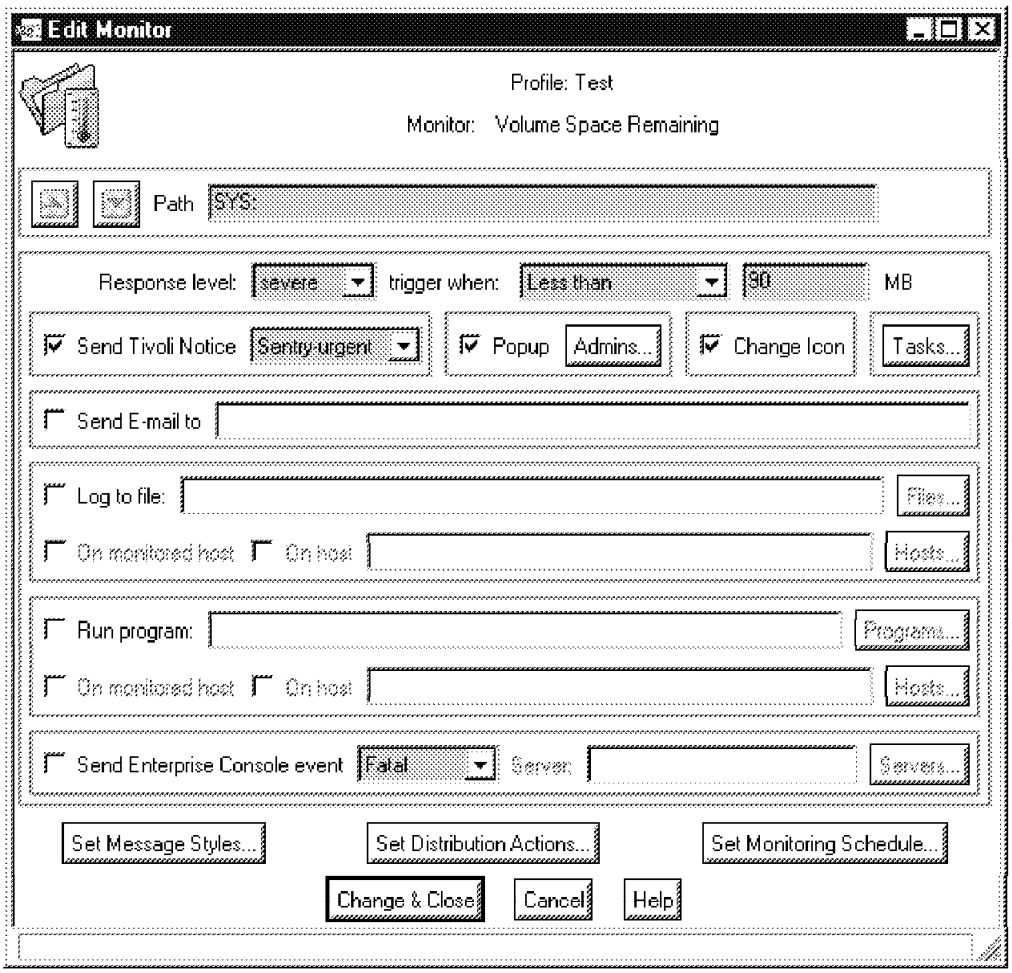


Figure 36. Defining Monitor Details

4. Click on **Set Monitoring Schedule** to define the interval at which to test the monitor. For testing purposes we chose to monitor at frequent (3 minute) intervals, but normally you would monitor less frequently.
5. Click on **Change and Close** to save the monitor. The resulting summary display is shown in Figure 37 on page 47. Note the indicator beside the new monitor entry. This warns you that the profile has not yet been saved. Select **File** and then **Save** from the menu bar to save the profile.

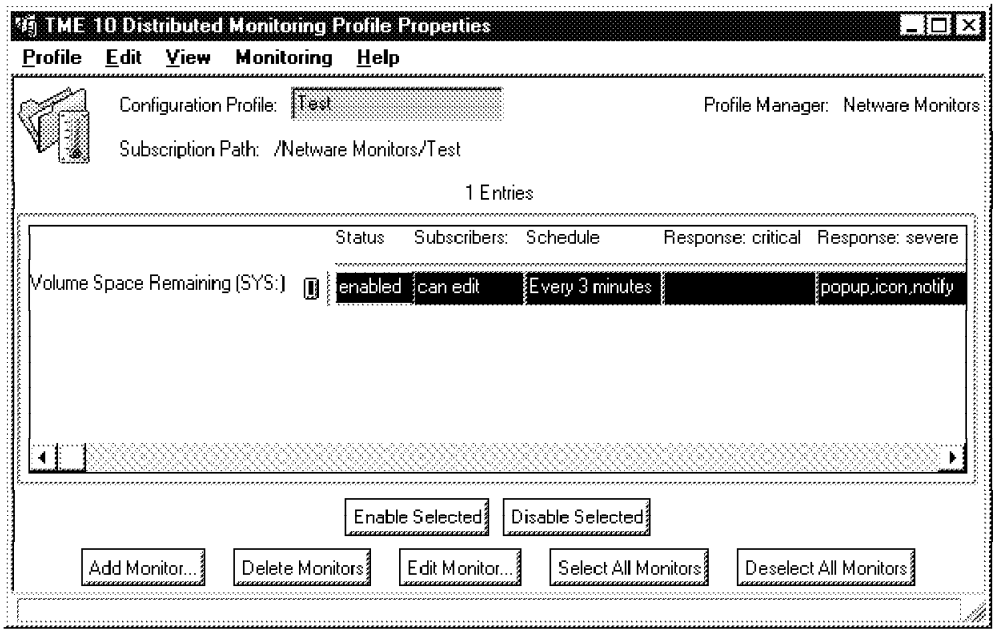


Figure 37. After Completing the Monitor Definition

5.3.3 Distributing a Monitor to a NetWare Endpoint

From the administrator's point of view, having created a monitor definition, distributing it is identical to any other Sentry profile distribution. Select **Profile** and then **Distribute** from the profile properties screen to distribute the profile with the default options. The status of the distribution is shown in messages on the TME desktop.

However, behind the scenes some special things are going on. The first time you distribute a monitor to a NetWare server there will be no Sentry engine (SENTRY.NLM) running to implement the request. Furthermore, the Sentry engine is dependent on *NetFinity Base Services*. NetFinity is used to provide the system instrumentation used by the NetWare monitors, rather than the Sentry engine having a lot of system-specific code built into it.

The first time a monitor is distributed to a NetWare server it has to install and start these pre-requisites. The way this is handled is as follows:

- The Sentry endpoint method, DOGENDPO.NLM is invoked as a normal downcall method. After the first invocation it will be in the method cache on disk at the endpoint, but the first time it is automatically downloaded by the spawner (LCFD.NLM). By default the endpoint method is cached in directory SYS:CACHEBINNW4TMESENTRY.
- DOGENDPO discovers that the Sentry engine NLM and the NetFinity NLMs it depends on are not installed. It invokes an upcall to retrieve them from the endpoint gateway. Note that although these modules are being downloaded and run in a very similar way to endpoint methods, they are not actually methods, but modules that are installed permanently on the system. Accordingly, they are loaded under the SYS:TIVOLI directory, not in the method cache (see the expanded directory tree shown in Figure 38 on page 48, compared to the original shown in Figure 22 on page 29).

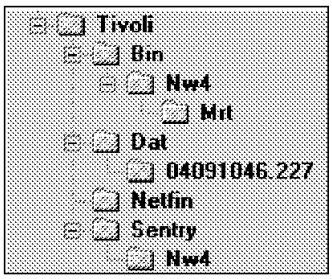


Figure 38. Tivoli Directory Tree after Distribution of First Monitor

If you watch LCFD.NLM while the distribution is taking place, you may see some of the modules being downloaded. One side effect is that the first monitor takes a lot longer to install than any subsequent ones. The size of the Sentry engine and NetFinity code is about 3 MB in total, which may take a significant time to load on a slow speed connection.

- We want the Sentry engine to be automatically reloaded when the server is restarted. DOGENDPO.NLM adds three lines to AUTOEXEC.NCF to achieve this. Near the start of the file it adds a search directory specification:

  ```
  SEARCH ADD SYS:TIVOLINETFIN
  ```

  At the end of the file it adds load commands for the NLMs:

  ```
  load sys:/tivoli/netfin/netfbase.nlm
  load sys:/tivoli/sentry/nw4/sentry.nlm
  ```

Finally, SENTRY.NLM and the NetFinity NLMs are loaded and the endpoint method to update the monitor list is executed. For subsequent monitor updates this is the only action performed when you distribute to the endpoint.

5.3.4 Monitor Responses from an LCF Endpoint

As you would expect, the responses generated by Sentry running on an LCF endpoint are presented in the same way as any other monitor. Behind the scenes the responses are being executed as upcalls to the endpoint gateway, which then invokes the kennel class methods on the TMR server to alert the administrator (setting icons, generating popup messages, etc).

In the case of our NetWare server free disk monitor, the administrator receives a pop-up when the free space drops below 90 MB, and the indicator icon shows the monitor state change. Figure 39 on page 49 shows the indicator and Figure 40 on page 49 shows the log that is displayed when you double-click on the indicator icon.

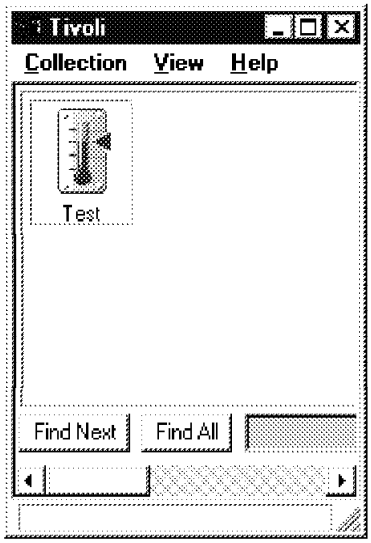


Figure 39. Indicator Shows Severe Error

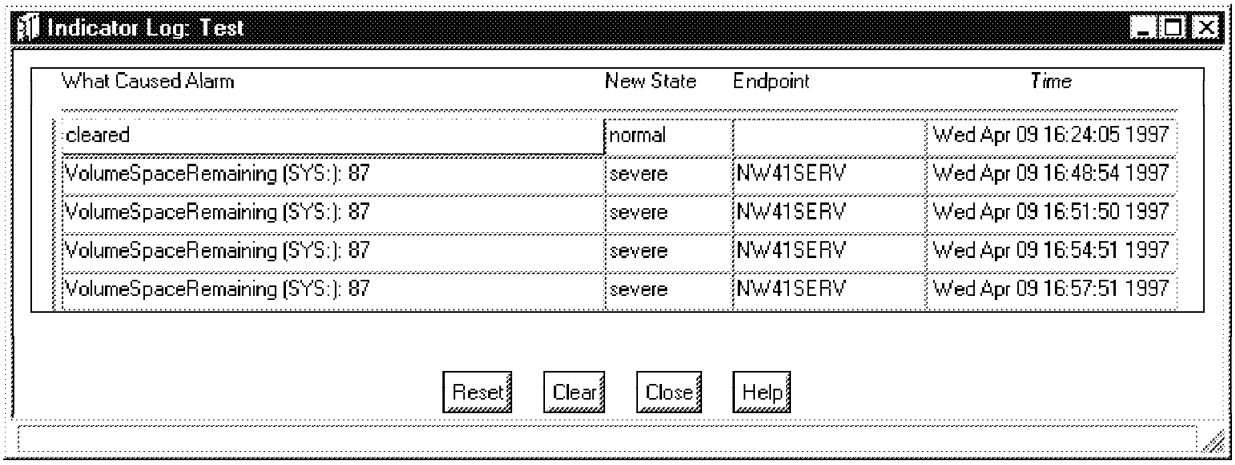


Figure 40. Indicator Threshold Log

Chapter 6. Data Collection and Graphing

TME 10 Distributed Monitoring is primarily an alerting mechanism. It sits quietly checking your systems and warning you, or taking action, when something is out of line. However, there are other uses for the kind of data that it monitors.

For example, if you have some kind of system problem you may want to view performance metrics regularly, so that you can watch out for errant patterns of behavior. You may also want to view statistical information spread over a period of time, to see the way that load varies over the course of a day for example. For more serious analysis and planning you may want to summarize data over weeks or months, to be able to spot trends and deal with potential points of failure in advance.

TME 10 Distributed Monitoring 3.5 has a data capture capability that logs the result of normal monitors to a file. It also provides a Web-based monitoring application that displays numerical data in a graphical form, updated dynamically. Concurrently, a new component of TME 10 Reporter is being developed which will read the TME 10 Distributed Monitoring log files and summarize the figures in its database.

6.1 Installing the Data Collection Function

The data collection and graphing function is delivered as an additional TME component on the TME 10 Distributed Monitoring CD. The product title is *Tivoli/Spider HTTP Daemon/1.0*. Install it as you would any other TME application. It has to be installed on all managed nodes for which you want to collect graphical data.

When you have installed the feature, a new process will be automatically started on each system. This is called *Spider* and it is a special purpose Web server. Do not be concerned if you already have a Web server running on a managed node. Spider does not listen on the default HTTP port (tcp/80), but on a dynamically assigned TCP port instead, so it will not conflict with existing applications.

Once you have installed the graphical monitoring component, the next step is to define some Sentry monitors to collect data, so that you can display it in graphical form.

6.2 Collecting Monitor Data

Starting the data collection facility is a very simple process. You create a new Sentry monitor within a profile in the normal way. In the monitor definition panel, select a response level of **always** and click on **Tasks** in the action section. Figure 41 on page 52 shows an example of doing this for a monitor that records the free space in a UNIX filesystem.

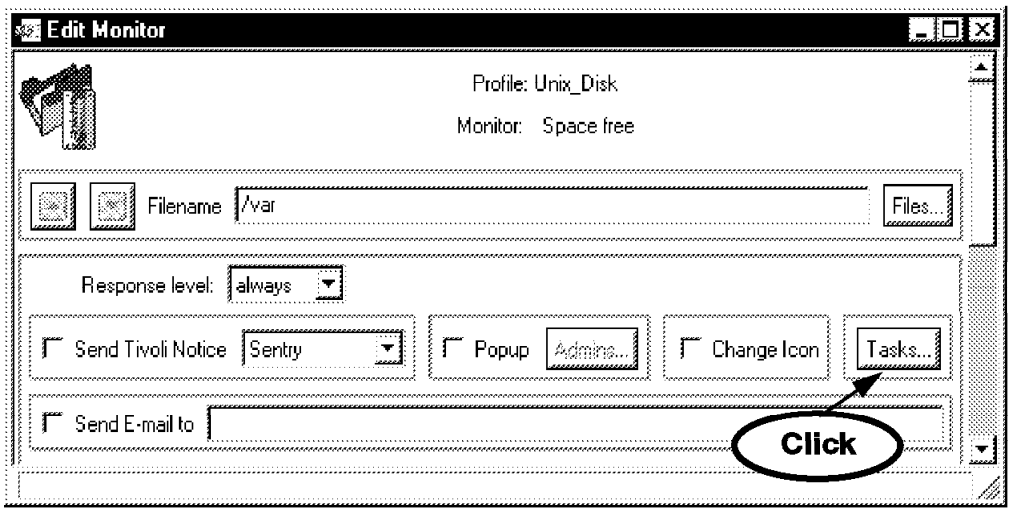


Figure 41. Defining an Always Response

In the Tasks dialog, select **Sentry Graphable Logs** as the task library and select task **Create Graphable Log** (see Figure 42).

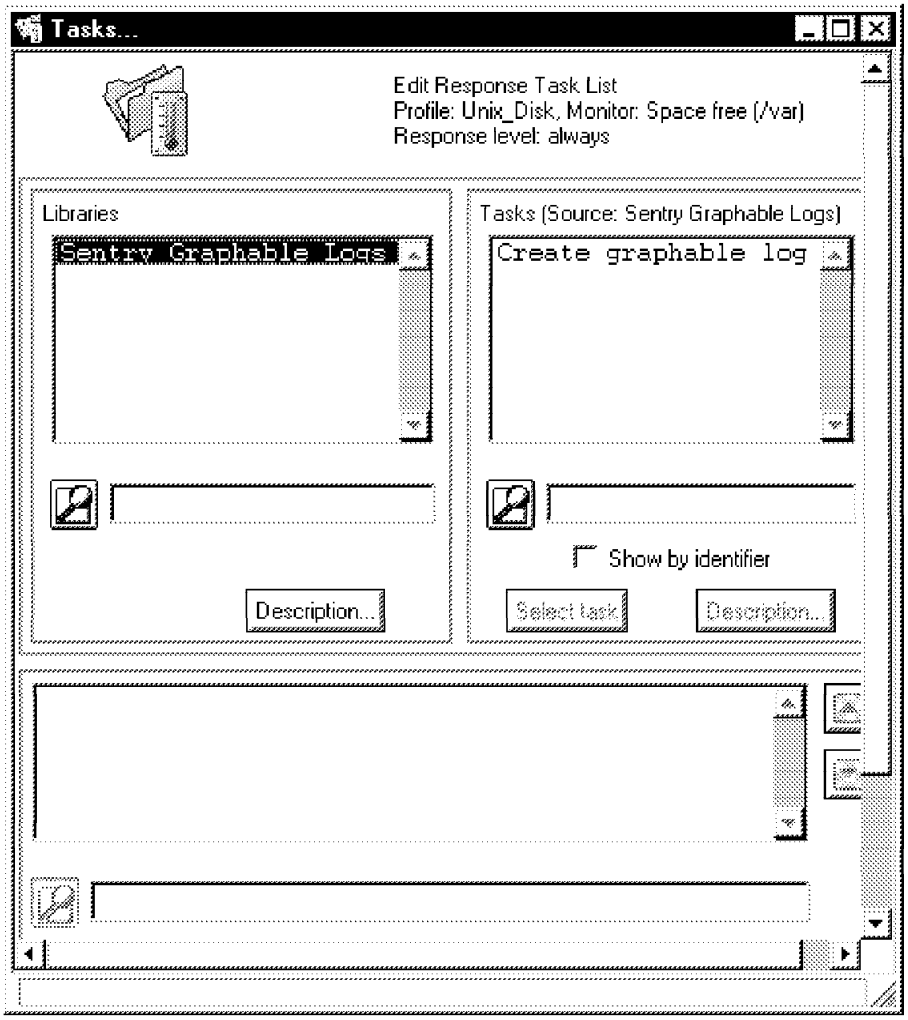


Figure 42. Defining the Logging Task

When you define the graphable log task you will be prompted to specify how many lines of data it should keep. The default is 1000. In most cases this will be large enough. For example, if you are collecting at ten minute intervals, and intend to archive the logged data daily you only need 144 data values.

The data files are stored in a directory on the managed node where the Sentry engine is running, under the Tivoli database directory:

```
$DBDIR/.sntglog/<Systemname>/<Profilename>/<Monitor_type><ObjID>
```

Where:

- $DBDIR is the Tivoli database directory on the managed node
- <Systemname> is the name of the system where the data was collected (usually the system where sentry_engine is running, but it could be another system in the case of a proxy monitor, for example).
- <Profilename> is the name of the TME 10 Distributed Monitoring profile containing the monitor. Unix_Disk in our example above.
- <Monitor_type> is a string identifying the type of monitor.It contains the name of the monitoring collection concatenated with the internal name of the monitor itself. In the example above it is Unix_Sentry-diskavail.
- <ObjID> is an identifier containing an oserv object ID number.

Within the directory are two files, info which contains details of the monitor and log which contains the data itself.

6.2.1 Log File Size Considerations

The log files are not very large, so space should not be an issue. Each data point in the file consists of a 8-digit time stamp, the monitor value and the text of the monitor status (critical, severe, warning, or normal). To take our disk space monitor above as an example, the data value is a two digit number followed by a decimal point and five decimal digits. The monitor status is usually normal, so the total size of each entry is 8+8+6, or 22 bytes. Even a 1000-entry log file will not be bigger than 22 KB.

6.2.2 Collecting Non-Numeric Data

Normally the data you are interested in collecting is numeric data, for graphing or statistical purposes. However, there is nothing to prevent you from using the data collection task on a string monitor if you wish. You could use this to capture status information, such as whether a daemon is up or down, for example. We show an example of extracting daemon status information from a log file in 6.5, “Extracting Logged Data from the Command Line” on page 59.

6.3 The Spider Web Server

The new graphical monitoring function in Sentry 3.5 is based on a network of Spider Web servers. Spider is stopped and restarted by the oserv daemon every time oserv stops and starts. You can also control it manually using the wstophttpd and wstarthttpd commands.

6.3.1 How Spider Works

Spider is a conventional Web server that handles HTTP get and post requests from any Web browser. However, it is unusual in the way it is integrated with the TME object request broker services. For example:

- It opens a dynamically-assigned TCP/IP port instead of listening on a fixed port number. It then registers this port with oserv. When you want to connect to Spider, you direct your browser to the normal oserv objcall port (tcp/94). oserv then responds with an HTTP LOCATION message, passing you to the Spider port.
- It uses system authentication services to validate your user ID and password, which it demands using the normal HTTP Basic Authentication challenge mechanism (sometimes called HTTP *security realms*). It then uses your credentials to authenticate you as a TME administrator.
- It provides CGI programs for executing TME commands and method calls as a result of HTTP requests, under the authorization roles assigned to your administrator ID.

Figure 43 illustrates how Spider is related to oserv and how a user gains access to it.

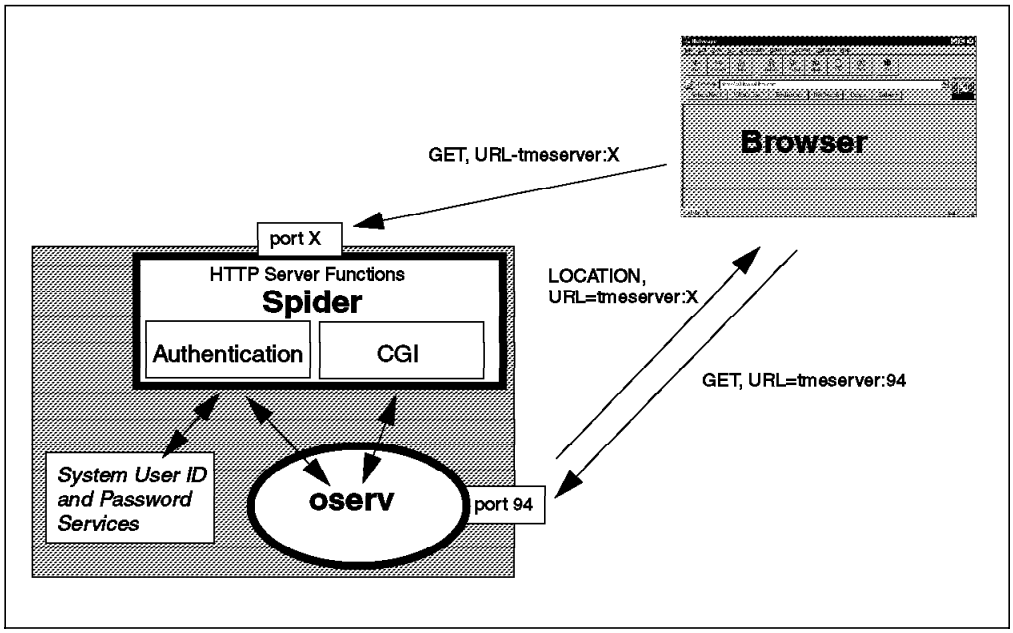


Figure 43. Spider Web Server Connections

6.4 Displaying TME 10 Distributed Monitoring Graphical Reports

To access the graphical reports, you must first connect to Spider on the TMR server. In the browser location field, select http://<tmr_server_name>:94. This will send an HTTP get request to oserv. If Spider is running, your session will be connected to it.

Figure 44 on page 55 shows the initial Spider page. This is a menu of applications. At the moment it only has the one entry in it, for TME 10 Distributed Monitoring, but as other TME applications make use of the Spider facility you can expect the list to grow.

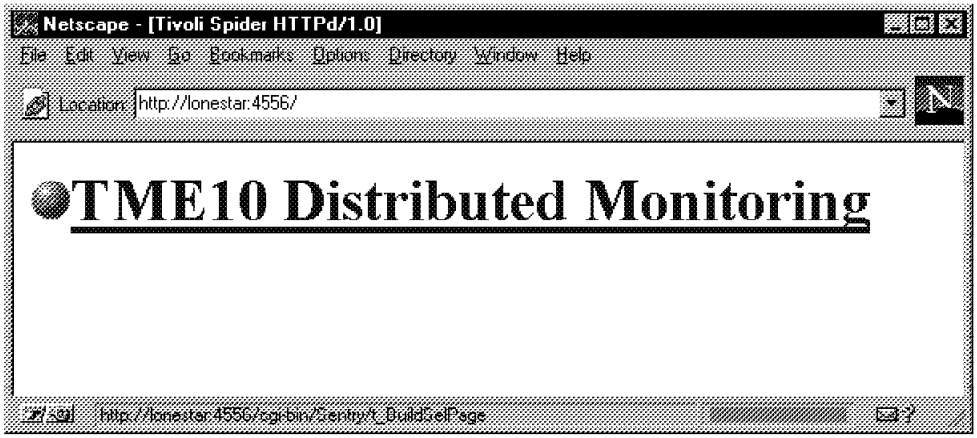


Figure 44. Initial Spider Menu

If you now click on the TME 10 Distributed Monitoring link, you will be prompted for a user ID and password (see Figure 45). Note that this is a system user ID on the TMR server. There are two things you should consider here:

1. This ID will be translated into a TME administrator ID. It is quite possible for someone to be defined as a TME administrator but *not* have a user ID on the server. You may have to create an additional user ID and modify administrator login definitions in order to give an administrator access to the graphical reports.
2. The mechanism for sending user ID and password, HTTP Basic Authentication, masks the values but does not encrypt them. If you are planning to allow people to access the graphs over the Internet, there is a risk that the IDs may be compromised.

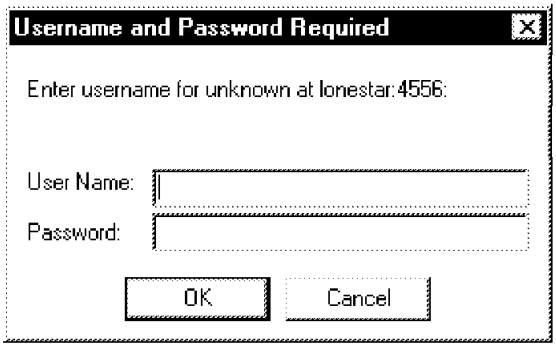


Figure 45. User ID and Password Prompt

Enter the ID and password and click on **OK**. A page will be loaded which contains a Java applet for defining the reports you want to see. Figure 46 on page 56 shows the report definition screen. It contains two list boxes. The one on the left defines the target domain, that is, the TMRs and the managed nodes within them. When you click on one or more nodes, the list on the right will show the monitors for which data has been logged.

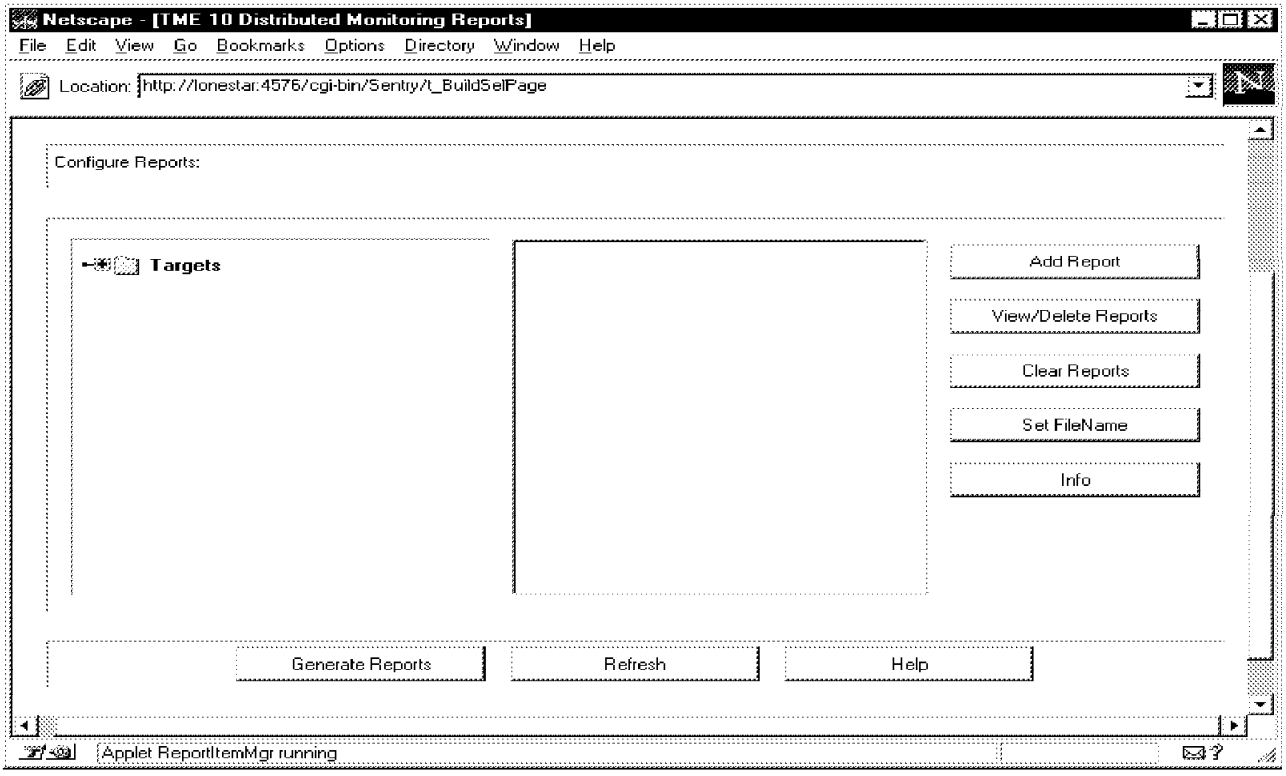


Figure 46. Report Selection Dialog

Open up the resource tree by clicking on the plus sign next to each icon. As you navigate down the tree it will look something like Figure 47.

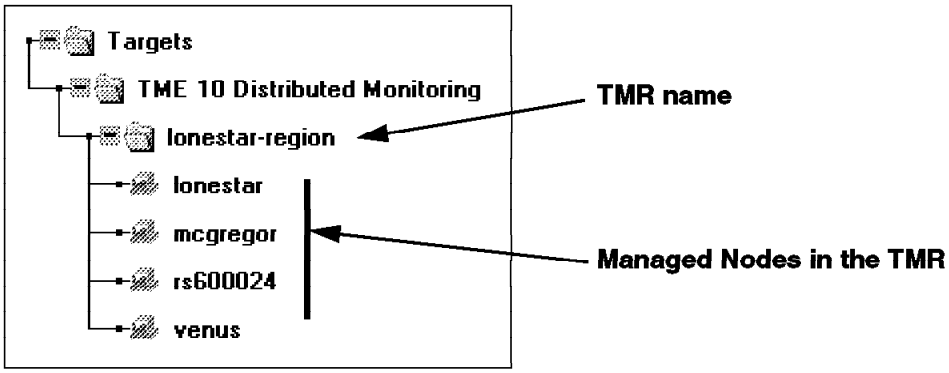


Figure 47. Monitoring Target Resource Tree

If you now select one of the target nodes from the tree, the Java applet requests Spider on that node to return a list of monitors with logged data. In fact, the request goes to Spider on the TME server, which finds out the TCP port for Spider on the target node and relays the request to it. A Java applet is not permitted to open network connections to any host other than the one it was loaded from.

Figure 48 on page 57 shows an example with three nodes selected and two monitors listed in the right hand box.

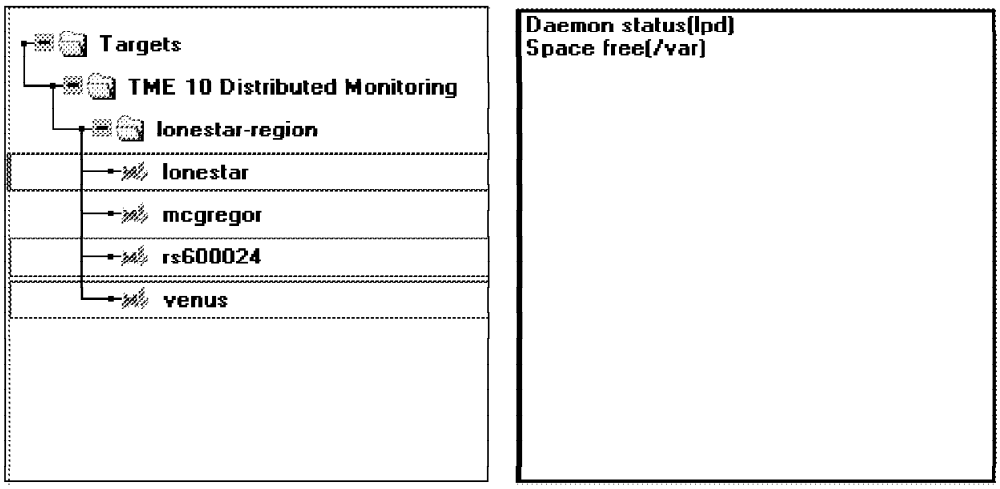


Figure 48. Selecting Targets to Display Collected Data

6.4.1 Defining the Reports

Now you can add graphical reports. You can have up to four graphs displayed at a time and you can have combinations of multiple monitors and multiple nodes on each graph. To create a graph:

1. Select the combination of nodes and monitors that you want to see. We selected two nodes and the Space free(/var) monitor.
2. Click on **Add Report**.

The screen shown in Figure 49 will pop up. Select the options you want and click on **Add/Close**.

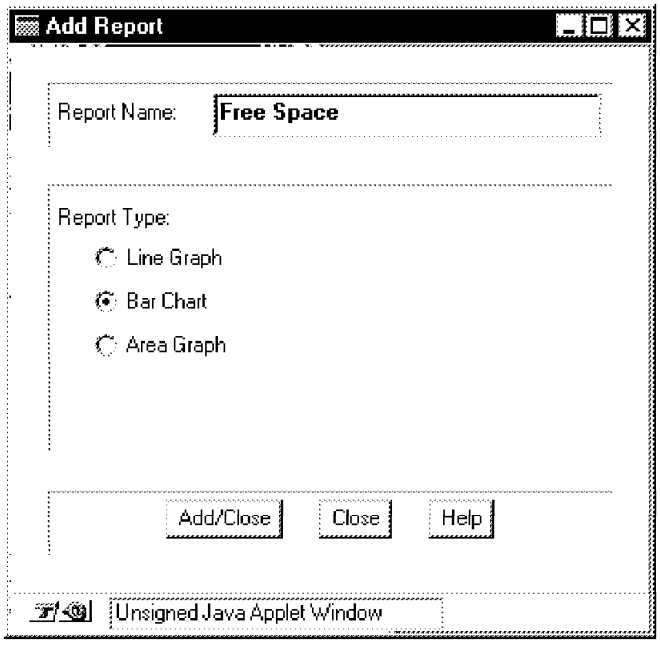


Figure 49. Report Options Screen

You will return to the main report selection screen. Select other combinations of nodes and monitors if you want to add more reports to the output.

When you have created the set of graphs that you want, click on **Generate Reports**. A new browser window will start up, containing a number of Java applets. Figure 50 on page 58 shows our example, a bar chart of space free in the /var filesystem for two UNIX machines.

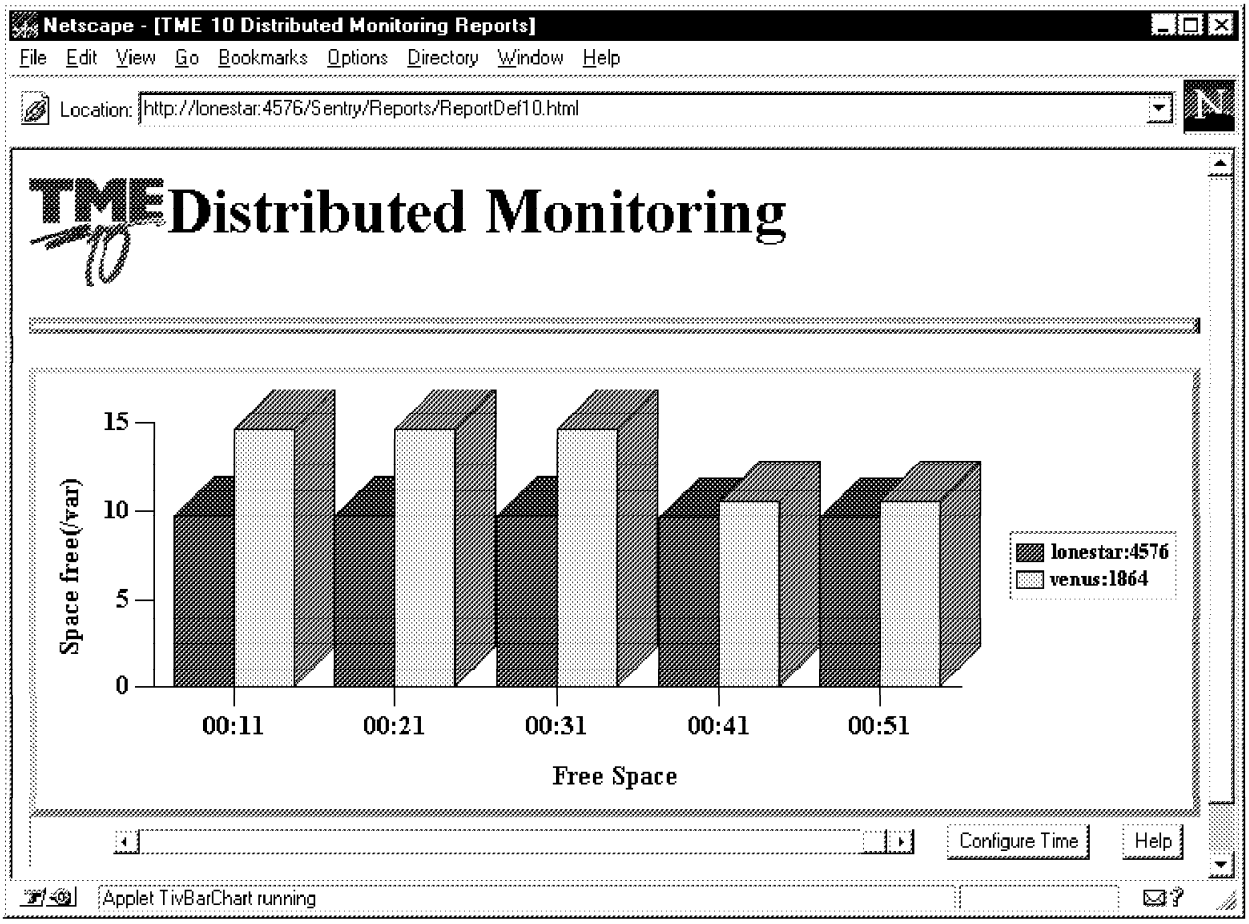


Figure 50. Graphical Reports in Action

You can use the scroll bar to look at data collected prior to the range of times shown on the x-axis. By default the graphs show the most current figures and update automatically when new data is logged.

When you first display the graph you may find that the information displayed is not what you expect. There are two things which can cause this:

1. When the graph is first presented, it will display eleven values at one minute intervals by default. If your monitor collects at larger intervals, the data shown will be from the last few monitored values, projected over the ten minute time range represented by the graph.
2. The timescale in the graph uses the timezone of the browser, not the data source. If you are using a desktop operating system, such as Windows 95 or Windows NT Workstation, you may not have needed to set the timezone before, in which case it will default to GMT.

Both of these problems can be corrected by clicking on **Configure Time**. You can also control the start time shown in the display and the number of intervals displayed using this dialog (see Figure 51 on page 59).

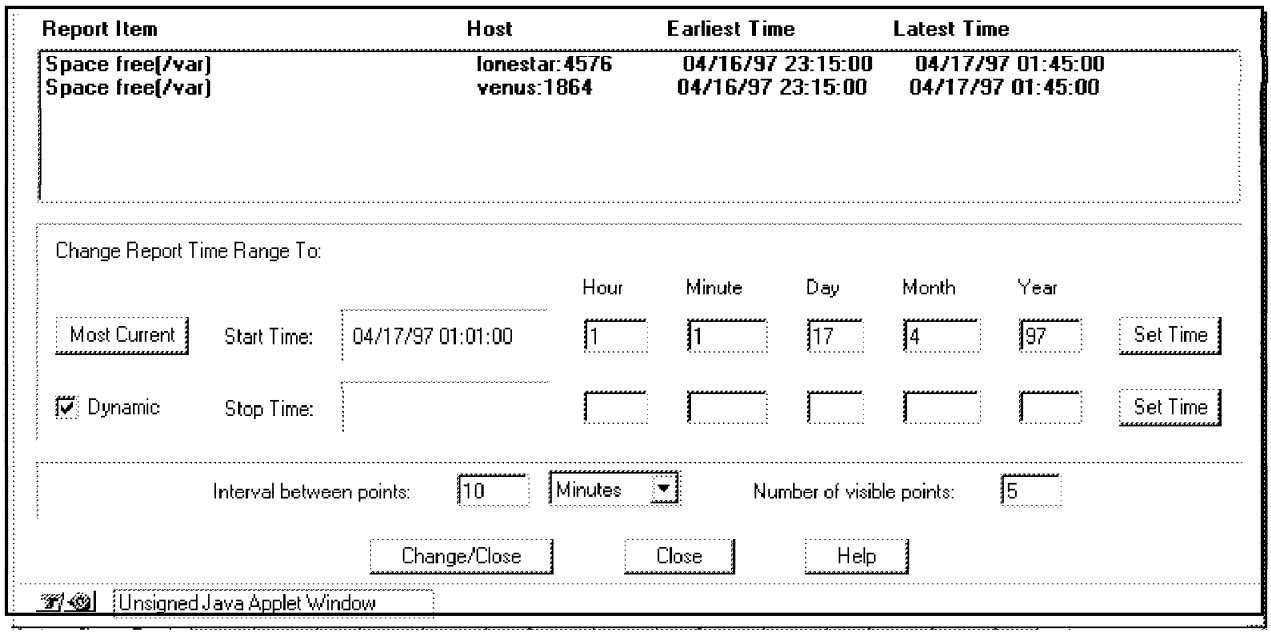


Figure 51. Time Configuration Screen

6.5 Extracting Logged Data from the Command Line

We described how the data collection task writes its log files in 6.2, “Collecting Monitor Data” on page 51. If you only want to show this data in a graphical form, the Web-based mechanism is very effective. However, you may want to extract the collected data and use it as input for some other processing, or load it into a database or a spreadsheet.

Data is logged as a single stream of bytes, containing the time stamp, value, and monitor status concatenated together. This is not particularly easy to parse into useful records. Fortunately there is a program, wgdread**,** which will format the records for you. It is not documented, but you can get a good understanding of how it works by looking at CGI scripts in the graphical monitoring application that use it.

The syntax is as follows:

```
wgdread [-r]|[-y]|[-u]|[-s [-b <min>] [-e <max>] [-c <count>] logdir
```

-r Returns the range of x values (times) of the records in the log file.

-y Returns the range of y values of the records in the log file.

-u Returns the units in which the y values are measured.

-s Prints out the logged values for the range of times bounded by <min> and <max> to a maximum of <count> lines.

logdir This is the directory containing the info and log files for the monito.r

As an example, Figure 52 on page 60 shows the use of wgdread to display part of a log file containing the status of a daemon. Note that this is created by a string monitor, so it would not be accessible using the graphing facility.

```
 >pwd
/var/spool/Tivoli/venus.db/.sntglog/venus/Unix_Disk
>
>ls
Unix_Sentry-daemon_7e1080755863.2.7
Unix_Sentry-diskavail_6e1080755863.2.7
>
>wgdread -r Unix_Sentry-daemon_7e1080755863.2.7
"Unix_Sentry-daemon_7e1080755863.2.7" 861252060 861311220 normal warning
severe critical
>
>wgdread -s -b 861296700 -c 10 Unix_Sentry-daemon_7e1080755863.2.7
861296701 down normal
861296820 down normal
861296940 down normal
861297060 down normal
861297180 down normal
861297301 up normal
861297420 up normal
861297540 up normal
861297660 up normal
861297780 up normal
```

Figure 52. Examples of Using wgdread

The time stamps in these records are not very recognizable. This is because they are in fact specified as *epoch* times, that is, they are a hexadecimal representation of the number of seconds since the beginning of 1970. The easiest way to convert these into something more meaningful is to use a perl program. Perl provides a number of built-in functions for manipulating time fields. Figure 53 shows a simple perl program that can be used as a filter for the output for the output of wgdread and Figure 54 shows an example of using it on the same data as in the previous example

```
 #!/usr/local/bin/perl

$line = <STDIN> ;

while ($line != "") {
        @inparts = split(/ /, $line) ;
        @tlist = localtime($inparts[0]) ;
        print ( $tlist[5],"/",$tlist[4],"/",$tlist[3]," ",$tlist[2],
                ":",$tlist[1],":",$tlist[0]," ",$inparts[1]," ",$inparts[2],"\n") ;
        $line = <STDIN>
        }
```

Figure 53. convert_times Perl Script

```
>wgdread -s -b 861296700 -c 10 Unix_Sentry-daemon_7e1080755863.2.7 | convert_times
97/3/17 13:5:1 down normal
97/3/17 13:7:0 down normal
97/3/17 13:9:0 down normal
97/3/17 13:11:0 down normal
97/3/17 13:13:0 down normal
97/3/17 13:15:1 up normal
97/3/17 13:17:0 up normal
97/3/17 13:19:0 up normal
97/3/17 13:21:0 up normal
97/3/17 13:23:0 up normal
```

Figure 54. Extracting Historical Data with Date and Time Conversion

Chapter 7. Database Monitoring Using TME 10 Distributed Monitoring

With TME 10 Distributed Monitoring 3.5 there are several new monitoring collections available. Two of them are for monitoring major RDBM systems, namely Oracle and Sybase.

In this chapter we give an overview of the monitoring collection for Oracle and show examples for its use.

7.1 Overview of the Oracle Monitoring Collection

The Oracle Monitoring collection that is delivered with TME 10 Distributed Monitoring 3.5 provides a set of basic monitors that can be used to monitor Oracle system parameters from the following categories:

- Oracle table space usage
- Oracle locks usage
- Oracle sessions usage
- Oracle CPU usage
- Oracle message traffic
- Oracle disk reads and writes

For table spaces, locks and sessions there are a couple of different monitors each, for example one that checks the locks already used and one that checks the locks still available.

All monitors use the Oracle SQL*Plus facility to query the information needed from the Oracle system tables. SQL*Plus provides a command line interface to the Oracle RDBMS server. The following figure shows this principle:

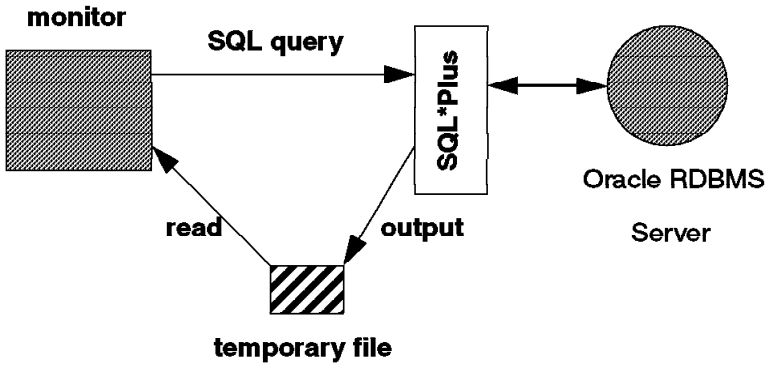


Figure 55. TME 10 Distributed Monitoring Talking to Oracle

Since SQL*Plus is an interactive query facility the ouput of a request is not directly usable by TME 10 Distributed Monitoring. Therefore, the output is first written to a temporary file that is then further processed. For example, the percentage value is calculated and then passed back to TME 10 Distributed Monitoring.

The monitors provide a good start to the monitoring of the RDBM system and are easy to use. However, if a more comprehensive management of the

database system is desired, for example, the automatic management of database users by TME 10 User Administration, other components of TME 10 need also to be considered.

This can be achieved either by using, for example, the Oracle monitoring collection and making extensions to TME 10 User Administration to handle Oracle users and to other TME 10 applications. Or, there are several TME 10 Plus modules available that provide comprehensive management solutions for specific database systems. These modules usually cover monitoring of the database system, event management, user management and other key functions.

7.2 Prerequisites

In order to use the Oracle monitoring collection you need to have at least the following software installed:

- TME 10 Framework 3.1 with the LCF Patch B
- TME 10 Distributed Monitoring 3.5
- Oracle RDBMS Server (We use Version 7.1.6.2.0)
- Oracle SQL*Plus

After installing TME 10 Distributed Monitoring 3.5 you can install the monitoring collection for Oracle. The following figure shows the install option:

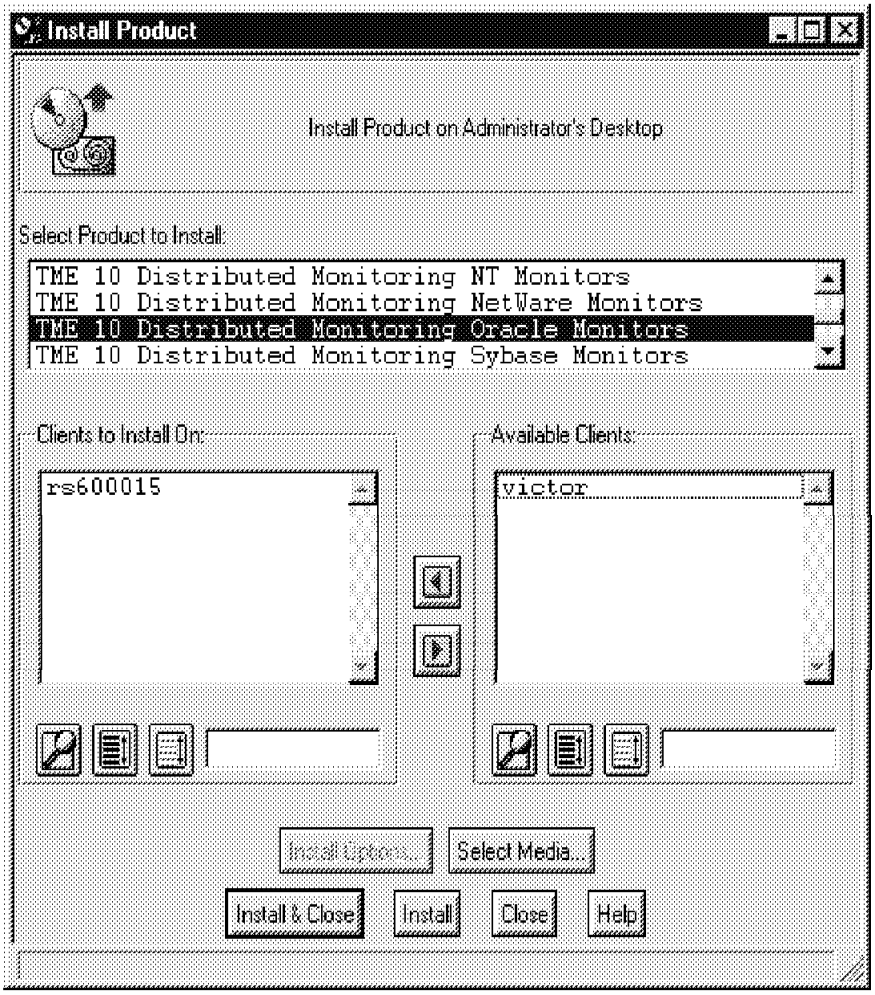


Figure 56. TME 10 Install Product Window

After the installation of TME 10 Distributed Monitoring Oracle Monitors the Oracle monitoring collection will be available.

The collection file will be located in:

```
/usr/local/Tivoli/bin/generic/SentryMonitors/oracle.col
```

To get an impression of how the monitors are implemented, you can type the following command:

```
mcsl -q Oracle
```

This command will produce a large text file. You will notice that all monitors are implemented as perl scripts that use SQL*Plus commands to obtain information that is then processed and sent to TME 10 Distributed Monitoring.

Before you start, you should make sure that your Oracle RDBMS server is installed and running properly.

If your database server is not running yet, start it by typing the following while being logged in as the Oracle database owner (usually, the user with the name oracle):

```
dbstart
```

This command will start the RDBMS server.

7.3 Producing Some Database Load

When you start testing the Oracle monitors it is likely that you will not operate in a database production environment where you have permanent load on the database system.

Maybe you even just installed Oracle and don't have any load at all. In the following we show a simple procedure that can be used to give Oracle some work to do while running the monitors.

After the installation of SQL*Plus (remember selecting to install the SQL*PLus demo tables during installation) you will have two sample files installed in the following directory:

```
/usr/local/oracle/sqlplus/demo
```

The above example assumes that /usr/local/oracle is your Oracle home directory. In this directory are two files, demobld.sql which will create some demo tables and insert values and demodrop.sql which will remove the demo tables from the system. We add another script select.sql that contains just the following lines:

```
select * from emp;
quit
```

To produce a constant load we can now type the following on the AIX command line:

```
cd /usr/local/oracle/sqlplus/demo
while true
do
sqlplus sys/oracle @demobld
sqlplus sys/oracle @select
sqlplus sys/oracle @demodrop
done
```

The above lines will add the demo tables, query one table, then remove the table again and start all over. This can be used to create some disk reads/writes etc.

Whenever you want to produce some database load, type the lines shown above.

Note

The above example assumes that the password of the Oracle sys user is set to oracle. If this is different in your environment, you must replace 'oracle' with the appropriate password.

7.4 Creating an Oracle Monitoring Profile

The oracle monitors can be created like any other standard TME 10 Distributed Monitoring profile. Open a profile manager in which you want to create the new TME 10 Distributed Monitoring profile:

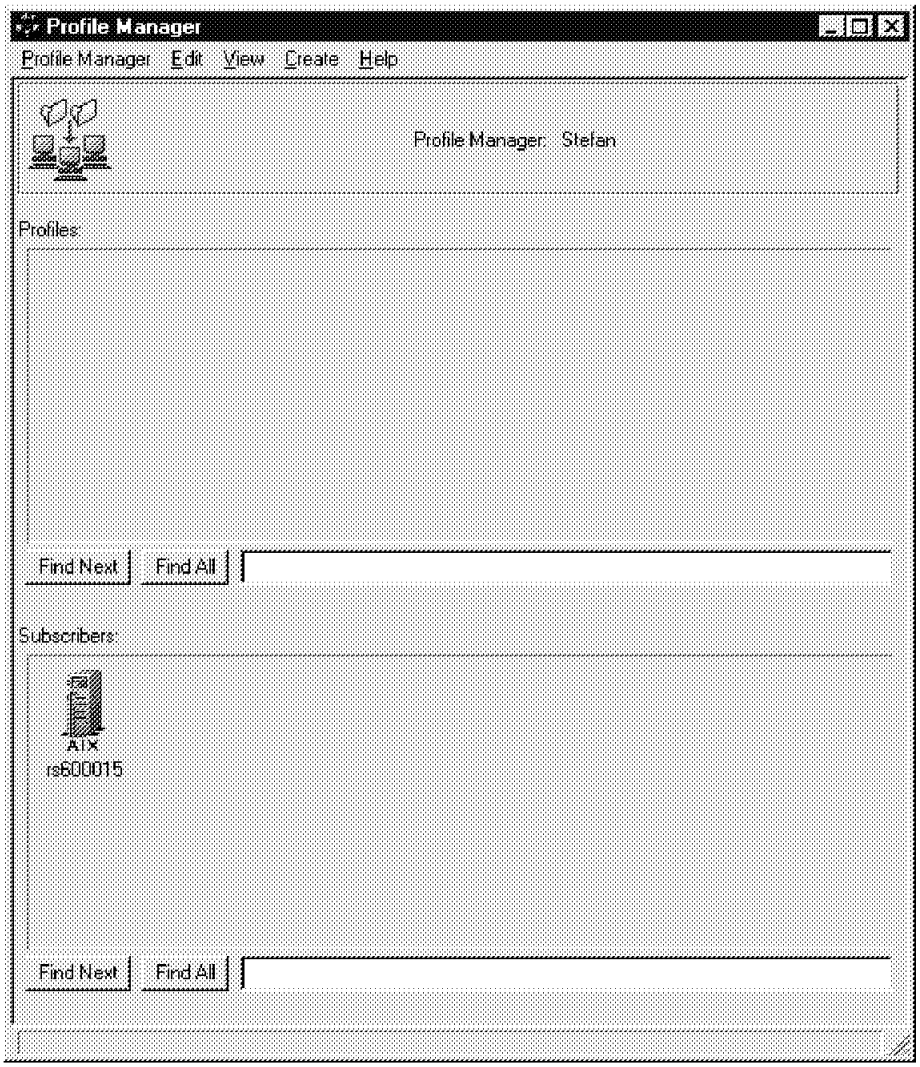


Figure 57. Profile Manager Window

Subscribe the managed node where Oracle is installed to the profile manager. Create a new TME 10 Distributed Monitoring profile by selecting **Create** from the menu bar and then **Profile...** from the pull-down menu.

The following window will appear:

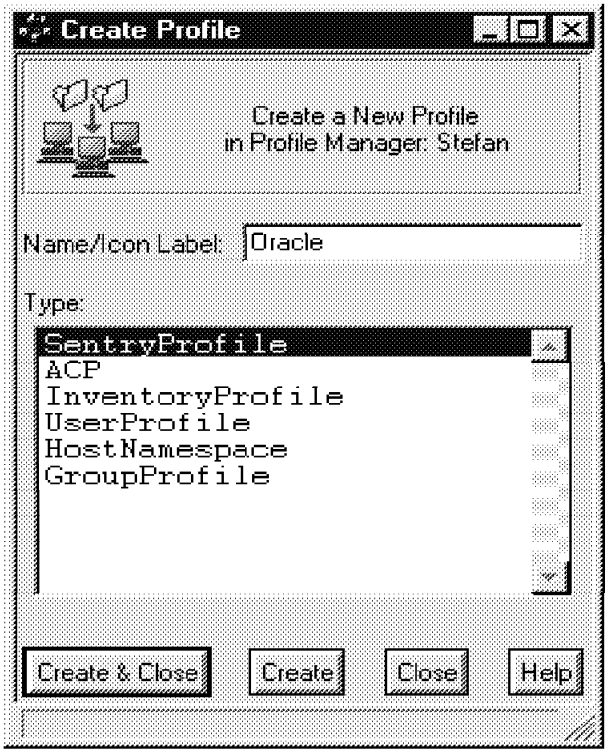


Figure 58. Create Profile Window

Enter a name for the new profile and select **SentryProfile** from the Type selection list. Then press the **Create & Close** button to create the profile. This will create an icon in the Profile Manager window representing the new profile.
Double-click on that item to open the profile:

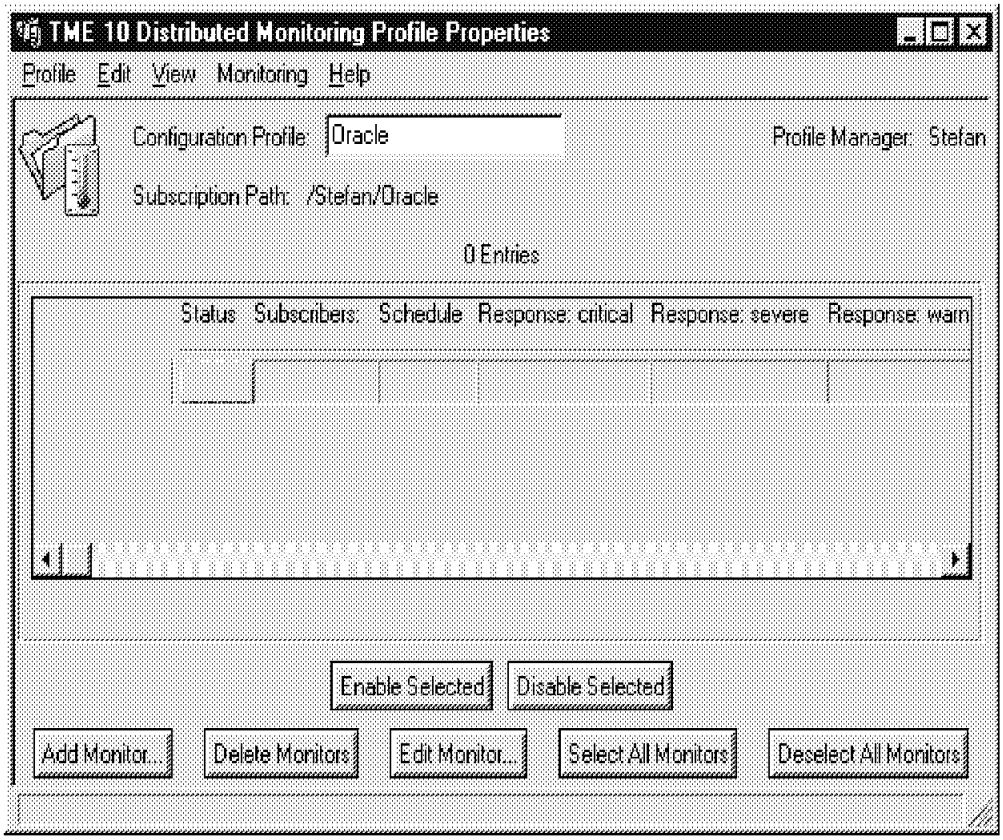


Figure 59. TME 10 Distributed Monitoring Profile

Click on then **Add Monitor...** button. The following window will appear:

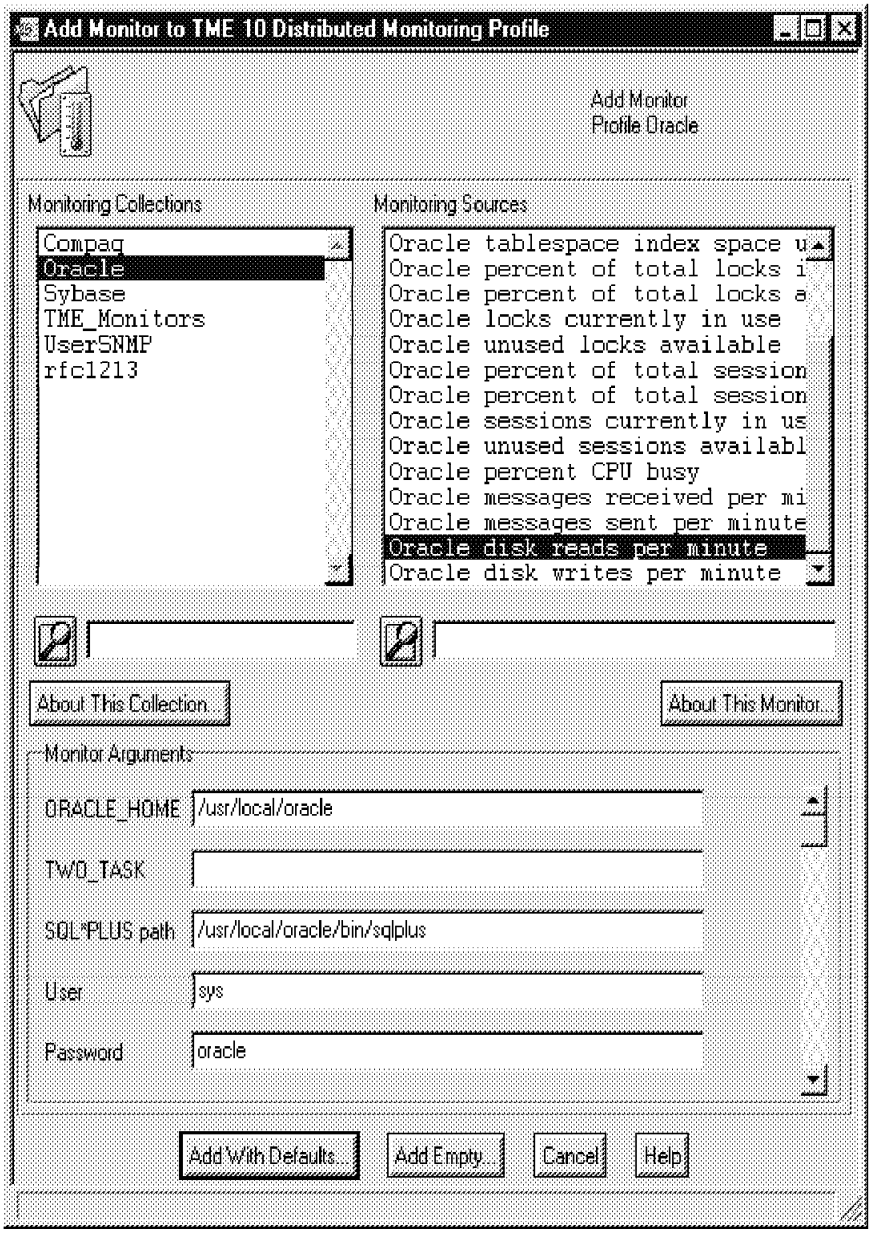


Figure 60. Add Monitor to TME 10 Distributed Monitoring Window

As you can see in the above window there is a new monitoring collection called Oracle available in the Monitoring Collections section. Double-click on **Oracle** to display the available monitors in the Monitoring Sources section.

Clicking on **Oracle disk reads per minute** will pop up the Monitor Arguments section. You need to supply the following values:

- ORACLE_HOME is the base directory where Oracle is installed, in our example /usr/local/oracle.
- TWO_TASK is the connect string for an Oracle two-task environment. In our case this is left blank, since our SQL*Plus client and the RDBMS server reside on the same machine.

- SQL*PLUS path is the path name of the sqlplus command. In our case this is /usr/local/oracle/bin/sqlplus. Do not confuse this with the /usr/local/oracle/sqlplus director.
- User and Password are the user ID and password used for accessing SQL*Plus. We use the Oracle sys user with a password of oracle. You specify the password for the sys user during Oracle installation.

Note

If you are not sure if the SQL*Plus user ID and password are correct, try starting the sqlplus command manually first and log on with the assumed user ID and password. If this works, specify the values in the monitor.

Select the **Add Empty...** button to go to the Edit Monitor window:

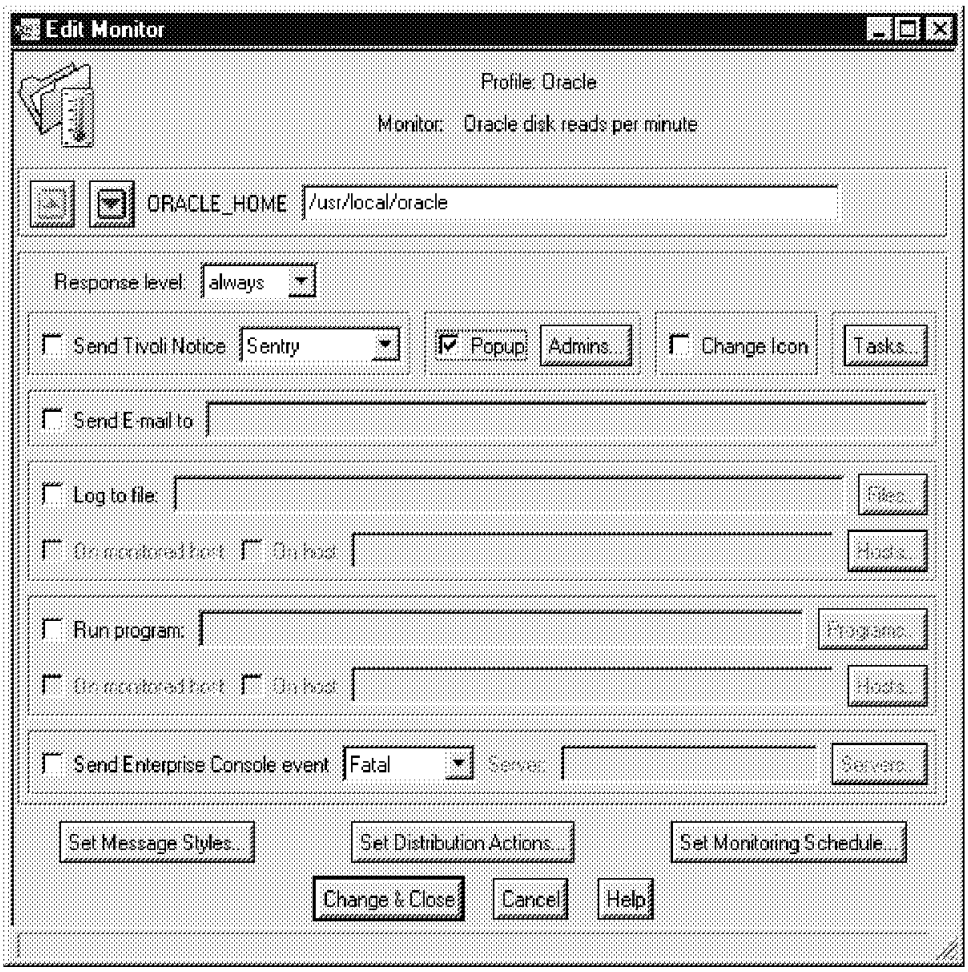


Figure 61. Edit Monitor Window

In order to see what kind of output the monitor produces it seems to be a good idea to just specify a trigger for the always response level first. This will trigger the specified action each time the monitor is run. We select the **Popup** check box and then click on the **Admins...** button to specify a TME administrator to receive the pop-up window. The we select the **Set Monitoring Schedule...** button and specify that the monitor should run every minute. You might also want to click on the **Set Message Styles...** button to set the output format for pop-up

messages, notice, etc. to get more detailed information for tracing. After doing so we select the **Change & Close** button. Back in the Profile Properties window we select **Profile** from the menu bar and then **Save** from the pull-down menu to save the profile. Then we close the Profile Properties window.

Back in the profile manager we distribute the new profile to the subscriber.

After a while, the following window will pop up informing us about the monitor probe:

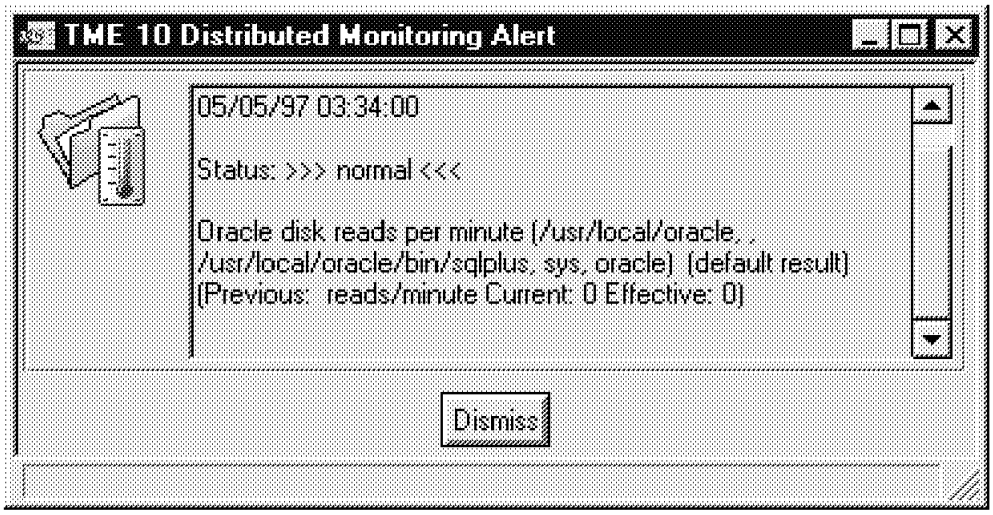


Figure 62. TME 10 Distributed Monitoring Alert Window

If you made a mistake when specifying the monitor, you should get an E.EXEC error message in the pop up window. For example, the following pop-up is created when specifying the SQL*Plus path /usr/local/oracle/sqlplus instead of the SQL*Plus executable /usr/local/oracle/bin/sqlplus:

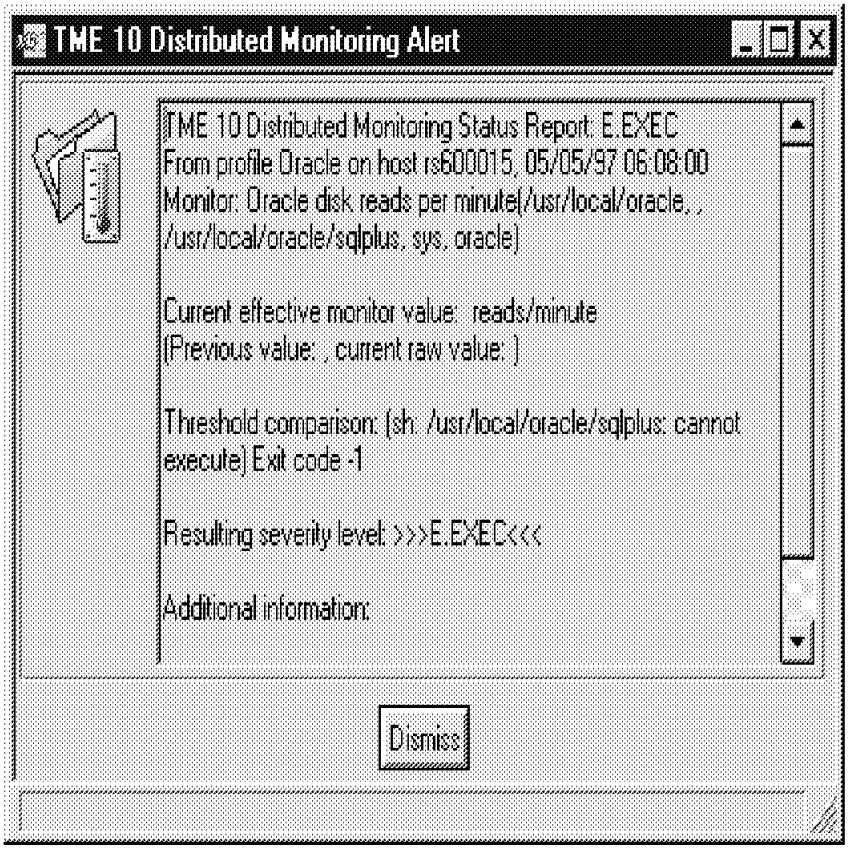


Figure 63. TME 10 Distributed Monitoring E.EXEC Error

With the Oracle disk reads per minute monitor you will notice that a file odb_reads.out is created in the /tmp directory.

Depending on the monitor, the following temporary files will be created:

Table 2. List of Temporary Files Depending on Oracle Monitor

Monitor Name	Temporary File
Oracle tablespace percent usage	odb_pct_spc_used.out
Oracle tablespace percent space available	odb_pct_spc_avail.out
Oracle tablespace total space allocated	odb_alloc_spc.out
Oracle tablespace total space used	odb_used_spc.out
Oracle index space used	odb_idx_size.out
Oracle percent of total locks in use	odb_locks_pct_used.out
Oracle percent of total locks available	odb_locks_pct_avail.out
Oracle locks currently in use	odb_locks_used.out
Oracle unused locks available	odb_locks_avail.out
Oracle percent of total sessions in use	odb_users_pct_used.out
Oracle percent of total sessions available	odb_users_pct_avail.out
Oracle sessions currently in use	odb_users_used.out
Oracle unused sessions available	odb_users_avail.out
Oracle percent CPU busy	odb_cpu.out
Oracle messages received per minute	odb_pkts_rcvd.out
Oracle messages sent per minute	odb_pkts_senbt.out
Oracle disk reads per minute	odb_reads.out
Oracle disk writes per minute	odb_writes.out

If you are experiencing problems with one of the monitors, it is always a good start to look in the /tmp directory to see if the temporary file for the monitor has been created. This file contains the output from the sqlplus command that the monitors use for further processing.

We experienced the problem that the temporary file was not created at all and the monitor would result in the following error:

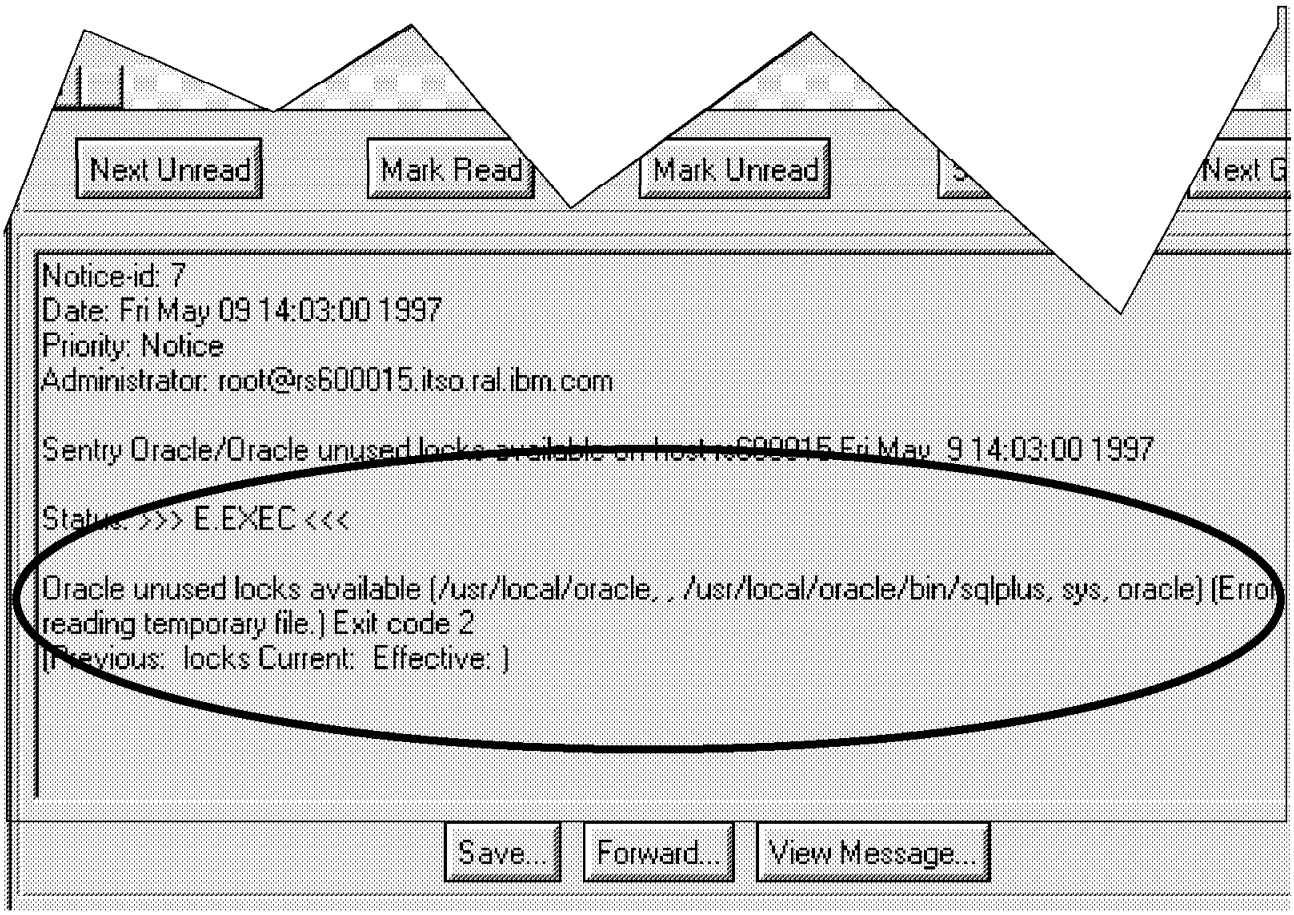


Figure 64. E.EXEC Error in Oracle Monitor

Note

In the above example, we set the monitor to send a message to the Tivoli notice board instead of popping up a window.

If this error occurs it is most likely to be caused by the sqlplus command that is called by the monitor failing.

The monitor directs the output of the sqlplus command to /dev/null when it is called, so a start to get an idea to what is happening is to look at this output. To be able to do that, you can move the original /dev/null file:

```
mv /dev/null /dev/null.bak
```

After the monitor has run, the output of the sqlplus command will be in a new file called /dev/null. In our example this looks like:

```
root@rs600015:/dev# cat null
SQL*Plus: Release 3.1.3.7.1 - Production on Fri May  9 15:17:00 1997
Copyright (c) Oracle Corporation 1979, 1994.  All rights reserved.
ERROR: ORA-02700: (Cnct err, can't get err txt.  See Servr Msgs & Codes Manual)
Enter user-name: Enter password:
ERROR: ORA-02700: (Cnct err, can't get err txt.  See Servr Msgs & Codes Manual)
Enter user-name: ERROR: ORA-02700: (Cnct err, can't get err txt.  See Servr Msgs
```

```
 & Codes Manual)
unable to CONNECT to ORACLE after 3 attempts, exiting SQL*Plus
root@rs600015:/dev#
```

In our example the command fails, because the Oracle environment is not set up correctly. To solve this problem, we have to just replace the path to the sqlplus command in our monitor with the name of a shell script in which we first set the environment and then call sqlplus.

The following figure shows the modified monitor:

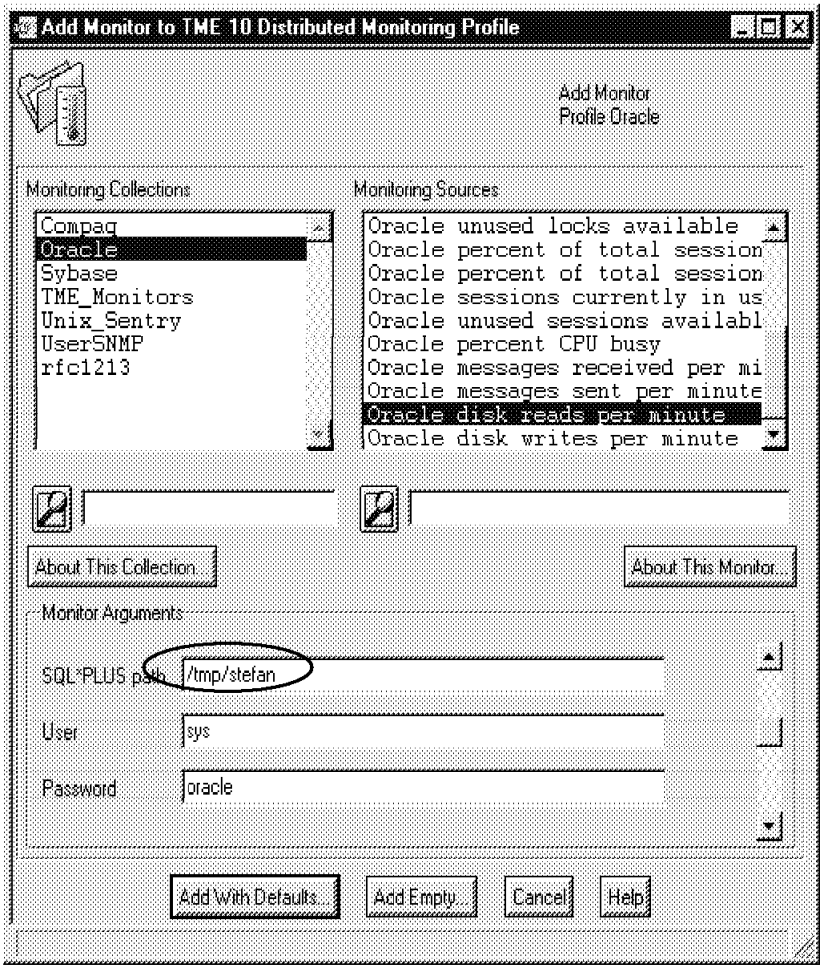


Figure 65. Add Oracle Monitor

The script /tmp/stefan looks like the following:

```
#!/bin/ksh
. /home/oracle/db_setup.sh
/usr/local/oracle/bin/sqlplus $* >/tmp/sqlplus.out 2>&1
```

It first calls /home/oracle/db_setup.sh to set up the correct environment for this Oracle system and the calls sqlplus, passing on the parameters from the monitor. The /home/oracle/db_setup.sh script for our environment looks like the following:

```
#!/bin/ksh
export ORACLE_OWNER=oracle
export ORACLE_HOME=/usr/local/oracle
export ORACLE_DOC=$ORACLE_HOME/docs
export PATH=$ORACLE_HOME/bin:/usr/lbin:$PATH
export ORACLE_SID=sid1
```

Both scripts must be set executable.

Now we can combine our monitor with the script to produce some database load presented in 7.3, “Producing Some Database Load” on page 64. We type the following lines and then watch the monitor:

```
cd /usr/local/oracle/sqlplus/demo
while true
do
sqlplus sys/oracle @demobld
sqlplus sys/oracle @select
sqlplus sys/oracle @demodrop
done
```

The following figure shows a notice sent to the Sentry-urgent notice group indicating the number of disk reads while the above script is running:

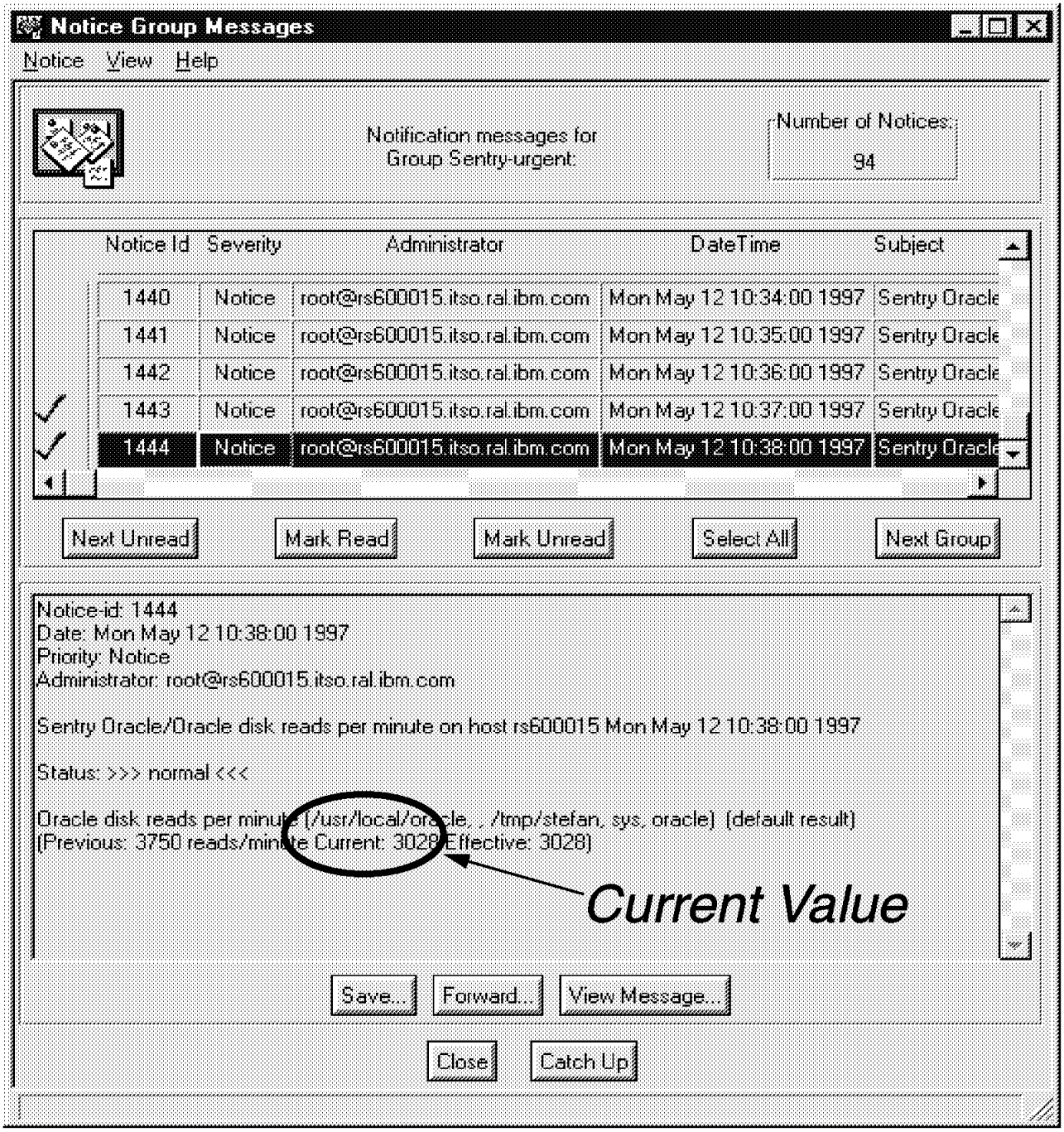


Figure 66. Oracle Disk Reads Monitor Under Load

7.5 Combining the Oracle Monitors with the Graphical Log Facility

A nice feature that can be combined with the database monitors is the graphical log facility that is also new with TME 10 Distributed Monitoring 3.5. This enables you to watch some of the database performance data, for example, disk reads and writes or CPU utilization in a graphical form.

In order to use this feature you need to have the Tivoli/Spider HTTP daemon installed.

The only addition you have to make to your monitor is to execute the task to create the graphical log each time the monitor is invoked. The following figure shows what to do:

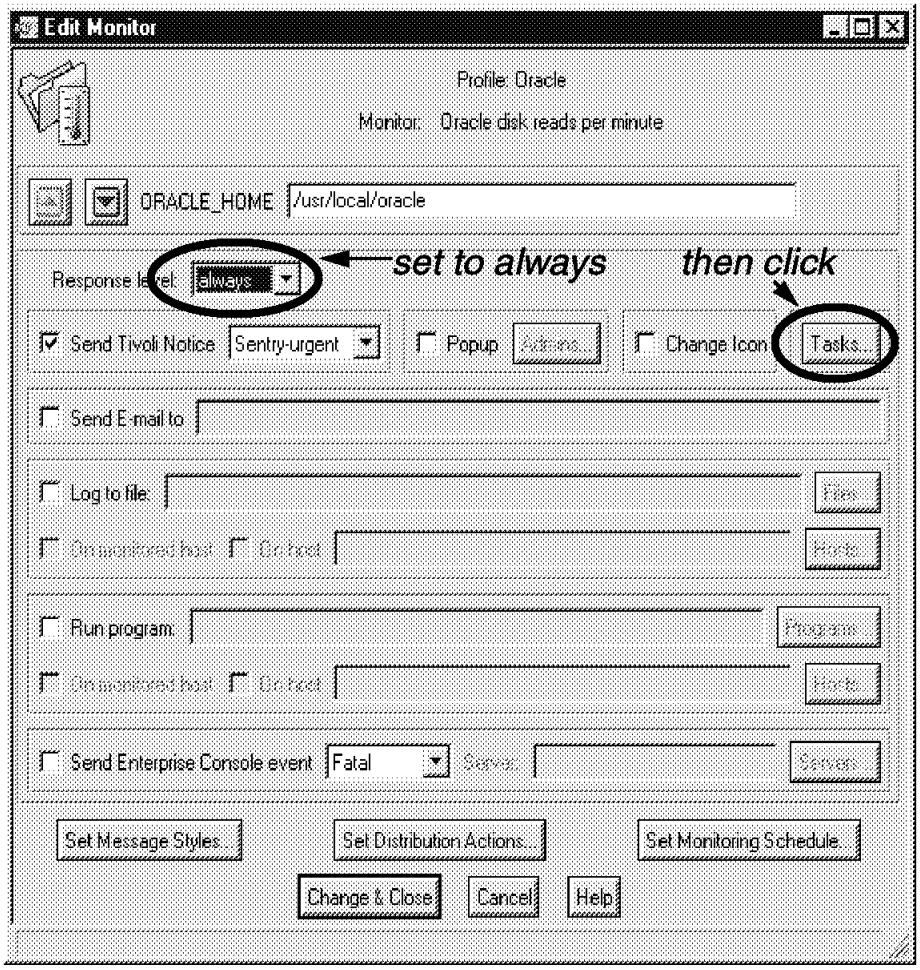


Figure 67. TME 10 Distributed Monitoring Edit Monitor Window

Set the Response level field to always and then click the **Task...** button.

This will pop up the following window:

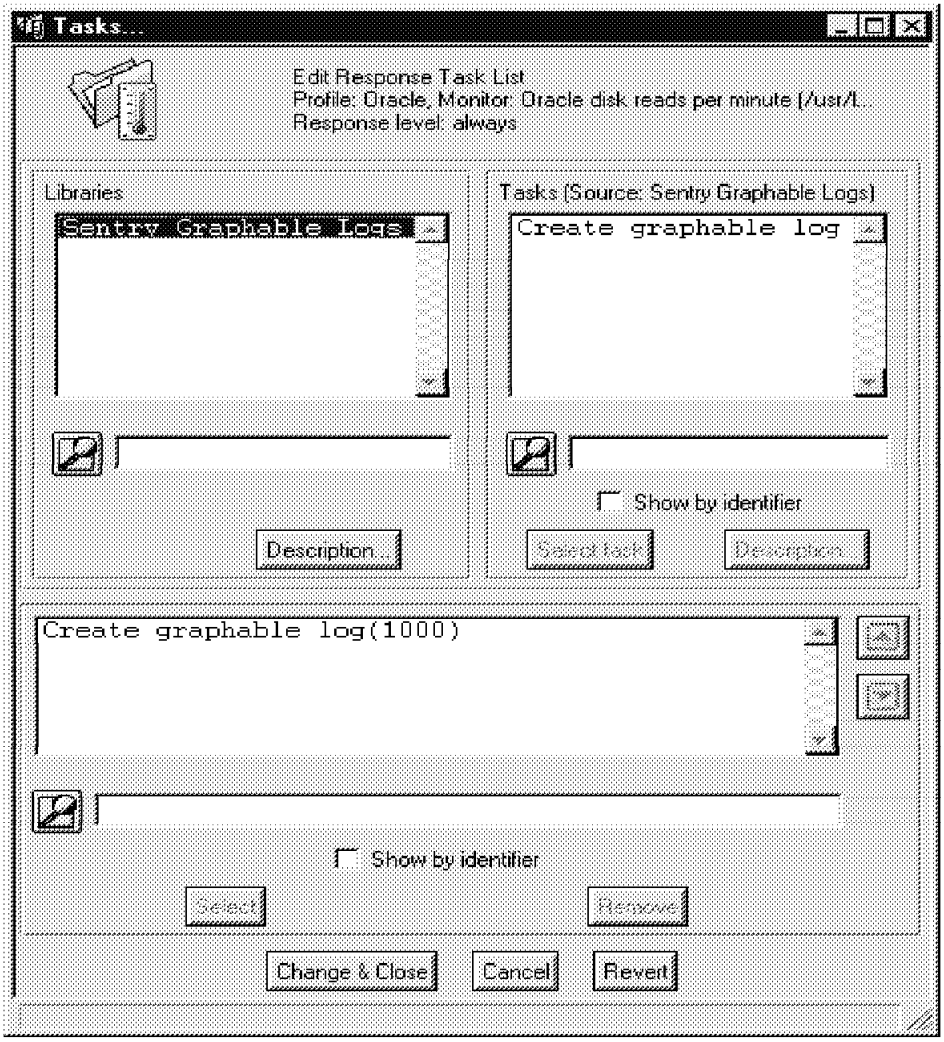


Figure 68. TME 10 Tasks Window

Select the **Sentry Graphable Logs** task library and then double-click on the **Create graphable log** task. You will be asked for the number of entries to keep in the log. In the above example we selected the default (1000).

When finished, click the **Change & Close** button and then save the modified monitoring profile. After the profile has been saved, distribute it again to the affected subscribers.

The results of the monitor will now be logged to a file. This file is stored under the /var/spool/Tivoli/.sntglog directory. In order to be able to watch the log using a Web browser, you must make sure that the Spider process is running on the machine where the TME 10 Distributed Monitoring server is installed.

If the Spider process is not running, start it by typing:

```
wstarthttpd
```

Now start your Java-enabled Web browser (we use Netscape) and point it to the following URL:

http:*hostname*:94

Where *hostname* is the name of the managed node where your TME 10 Distributed Monitoring server and Spider daemon are installed. In our case this is rs600015.

In the initial page click on TME 10 Distributed Monitoring. The system will ask you for a user ID and password to access the system

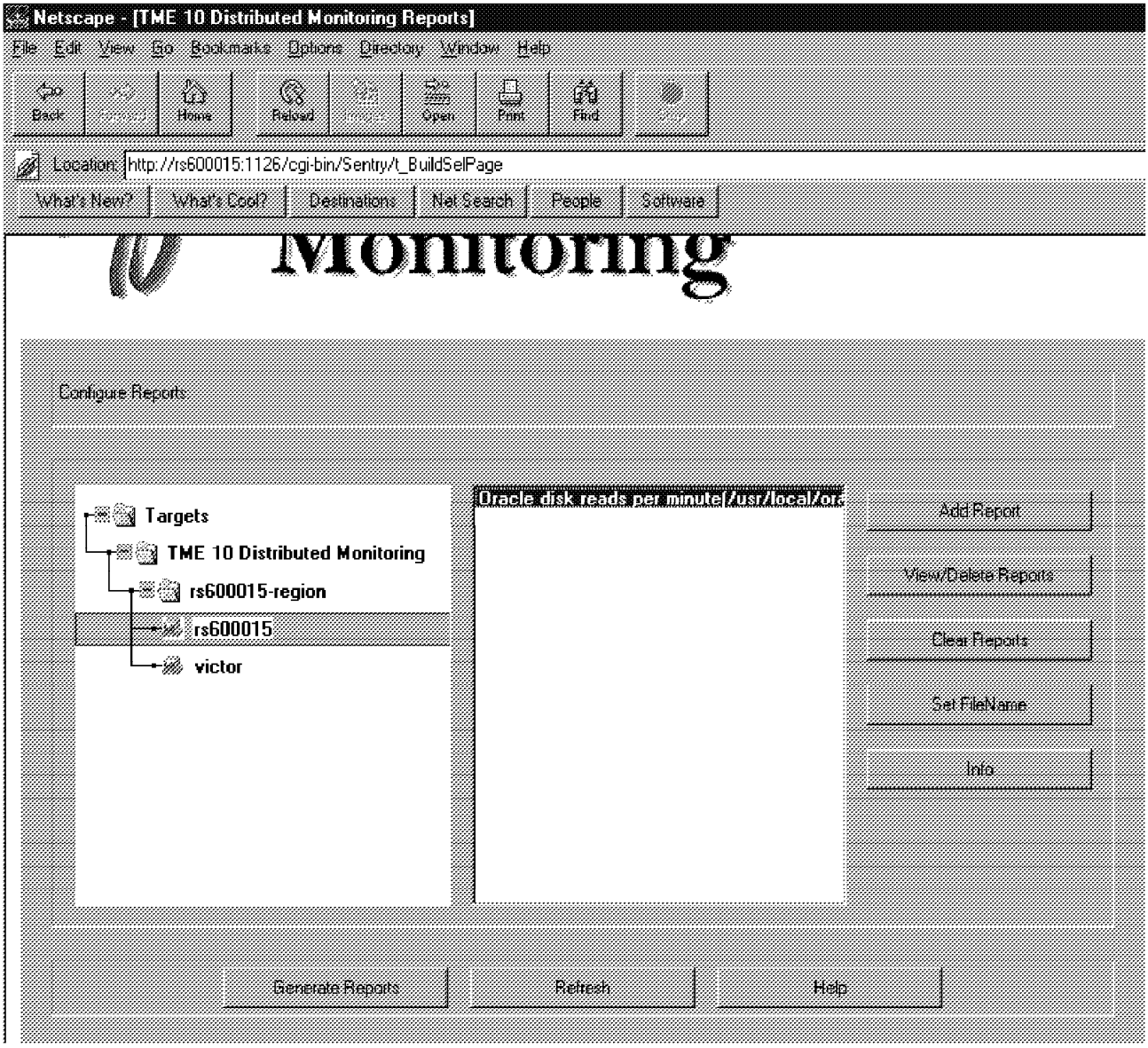


Figure 69. Web Interface for TME 10 Distributed Monitoring

Initially you will only see the **Targets** option. Double-click on that option to pop-up the related options until you get to the node where the Oracle monitor resides. In our case this is rs600015.

Clicking on **rs600015** will show the available monitors in the right window. Select the Oracle monitor and then click on the **Add Report** button. The system will ask you for a name for the new report and the style of the output graph:

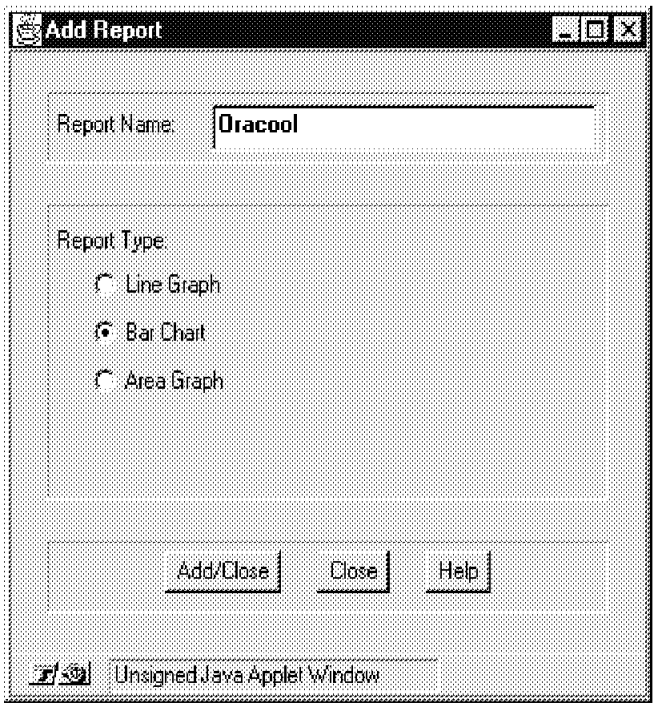


Figure 70. Add Report Window

Enter a Report Name and select a Report Type, then select the **Add/Close** button.

This will get you back to the window shown in Figure 69 on page 79. Click on Generate Reports and you will see a graphical output for the monitor.

The following figure shows an example of a graphical representation of an Oracle monitor:

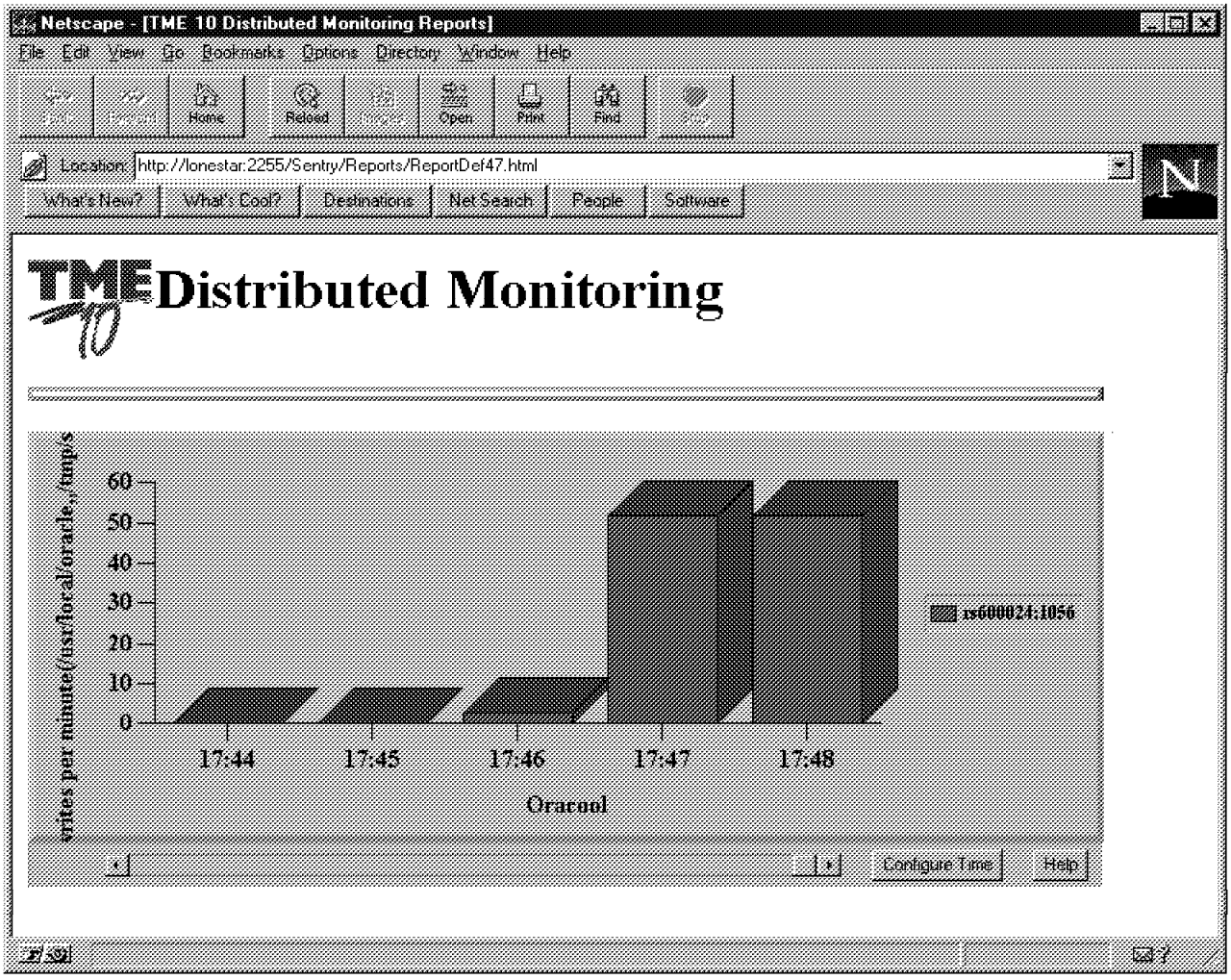


Figure 71. Graphical Representation of Oracle Monitor

The report shown above was generated from the Oracle disk writes per minute monitor that works in exactly the same fashion as the Oracle disk reads per minute monitor.

When the report was started (at 17:44 or 5:44 pm) the Oracle RDBMS server was idle. To produce some database load we then started the procedure shown in 7.3, “Producing Some Database Load” on page 64. Shortly after this we can see the number of disk writes increasing. When the procedure is stopped, the disk writes will decrease again.

If you want to see more than just five probes at a time you can click on the **Configure Time** button. This will pop up the following window:

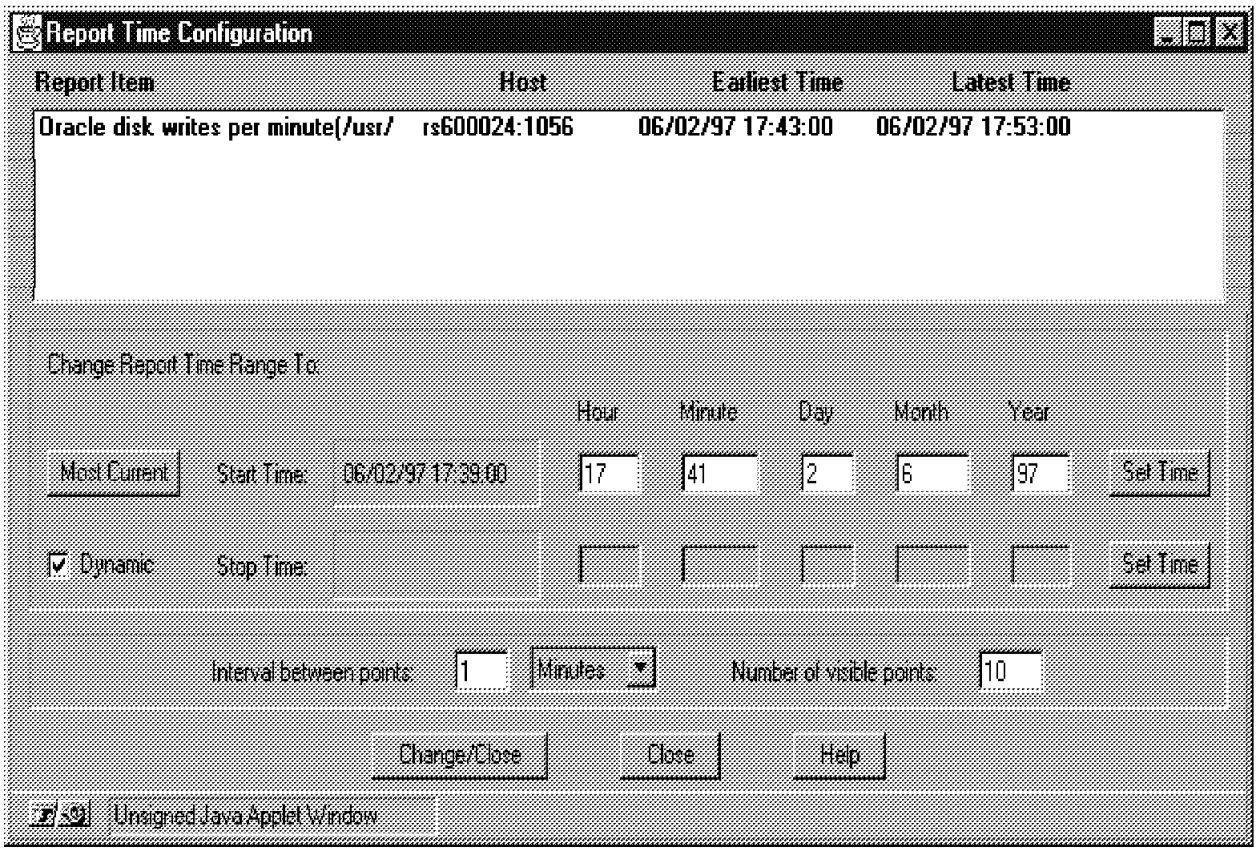


Figure 72. Report Time Configuration Window

Appendix A. Special Notices

This publication is intended to help systems specialists, planners and administrators to understand the capabilities of TME 10 Distributed Monitoring 3.5. The information in this publication is not intended as the specification of any programming interfaces that are provided by any Tivoli products. See the Programming Announcement for TME 10 Distributed Mionitoring for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (″vendor″) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have 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.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

NetView
AIX
OS/2
RS/6000
SystemView
NetFinity
System/390
AS/400

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.

Lotus and Notes are trademarks of the Lotus Development Corporation.

Tivoli, Tivoli Management Platform, TME and TME 10 are trademarks of Tivoli Systems Inc.

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 87.

- *TME 10 Cookbook for AIX*, SG24-4867
- *TME 3.0 NT - Automated Processes*, SG24-4793

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

These publications are also relevant as further information sources:

- *Tivoli Sentry Documentation Kit,* SK2T-6052

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, type one of the following commands:

  ```
  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, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO: type the following command:

  ```
  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.ibmlink.ibm.com/pbl/pbl

  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.ibmlink.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.

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Home Page (http://www.redbooks.ibm.com/redpieces.htm). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
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
- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **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.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Home Page (http://www.redbooks.ibm.com/redpieces.htm). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Index

Special Characters

Numerics

A

B

C

D

N

O

P

R

S

T

U

V

W

ITSO Redbook Evaluation

A First Look at TME 10 Distributed Monitoring 3.5
SG24-2112-00

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 redbook@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!)**

_____

_____

_____

_____

_____

This soft copy for use by IBM employees only.

Printed in U.S.A.