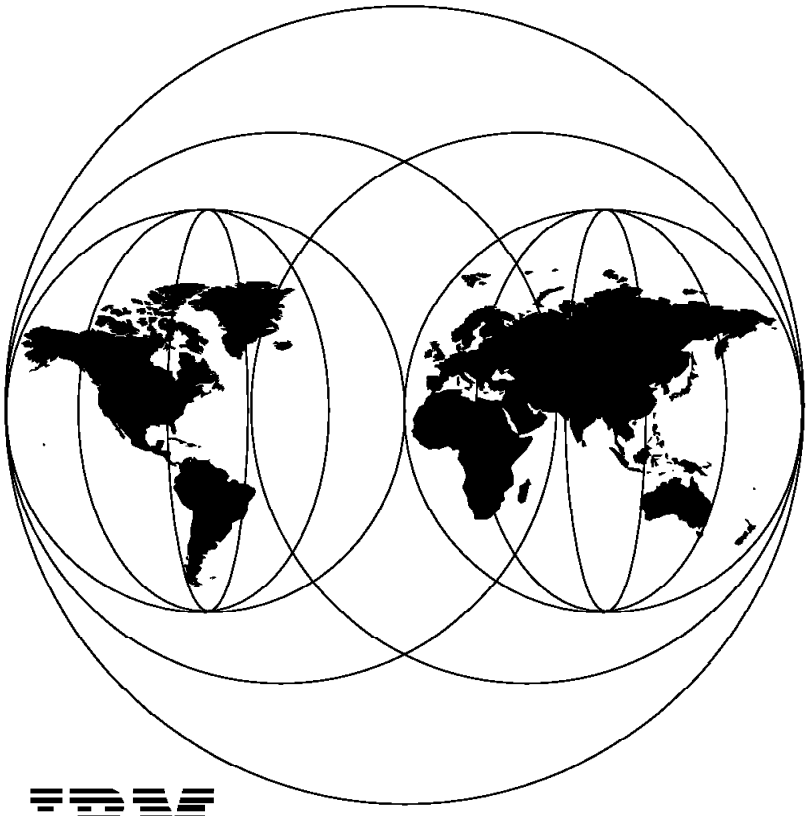
SG24-4936-00

# Migrating from Systems Monitor for AIX to TME 10 Distributed Monitoring

July 1997



International Technical Support Organization
Raleigh Center

International Technical Support Organization

SG24-4936-00

# Migrating from Systems Monitor for AIX to TME 10 Distributed Monitoring

July 1997

**Take Note!**

Before using this information and the product it supports, be sure to read the general information in Appendix C, “Special Notices” on page 143.

**First Edition (July 1997)**

**Take Note!**

Before using this information and the product it supports, be sure to read the general information under Appendix C, “Special Notices” on page 143.

This edition applies to TME 10 Distributed Monitoring Version 3.0.2 and 3.5 and to Systems Monitor for AIX Version 2.1.2.

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 describes the evolution of TME 10 Distributed Monitoring (previously known as Tivoli/Sentry), the Tivoli Management Environment application for monitoring the behavior of remote systems.

The redbook shows how to migrate from the previous SystemView monitoring application, Systems Monitor for AIX, and explains the different capabilities of the old and new approaches. It will help you position TME 10 Distributed Monitoring and TME 10 as a solution for enterprise systems management.

Numerous practical examples are used to illustrate the migration process and to show the capabilities of the latest version of TME 10 Distributed Monitoring.

All scenarios in this redbook are documented in a way that service providers can use the examples as a base for 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 Wojton and Paul Braun
Systems Management and Networking 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 153 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

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

- Send us a note at the following address:

  `redbook@vnet.ibm.com`

# Chapter 1. TME 10 Distributed Monitoring Overview

In this chapter we discuss the products that make up TME 10 Distributed Monitoring as they appeared before the Tivoli merger, now (July 1997), and forward into the future. We will limit the discussion to distributed management in open, heterogenous environments. The Netview/390 product is not discussed in this redbook.

## 1.1 Distributed Monitoring History

Before the Tivoli merger, IBM provided distributed monitoring capabilities with its SystemView product. For heterogenous environments, NetView for AIX and IBM Systems Monitor provided the tools and applications for distributed management based on the Simple Network Management Protocol (SNMP). SNMP is an IP based protocol and, although some SystemView applications implemented SNMP on other transports, Systems Monitor was limited to TCP/IP networks only. Systems Monitor was in three parts:

1. The System Information Agent (SIA), which provided system instrumentation and file monitoring functions.
2. The Mid-Level Manager (MLM), which provided SNMP polling, thresholding, automation and event filtering.
3. The System-Level Manager (SLM), which provided the MLM function locally on a system with the SIA also installed.

Meanwhile, Tivoli systems had developed the Tivoli Management Environment (TME), an implementation of the Common Object Request Broker Architecture (CORBA) optimized for systems management tasks. Built on the TME base platform are a number of applications for distributed system deployment, administration and monitoring. The primary monitoring application was called Tivoli/Sentry. Sentry used the framework to distribute system threshold definitions to monitoring code located on the target systems. The framework also provides facilities for alerting administrators about the status of monitors and executing automation tasks.

Apart from the different mechanisms used by the two products for control and alerting, there are also significant philosophical differences. Systems Monitor SIA is embedded into the system, directly accessing performance and status information and exposing it in the form of an SNMP MIB. Sentry, on the other hand, is designed to tread softly on its host system. It gets the information for its monitors from commands and interfaces provided by the system and applications, instead of going under the covers to get the data. This approach may be less efficient than the embedded method, but it is much less prone to failure due to system changes.

Figure 1 on page 2 shows the relationships between the different components of Systems Monitor and Sentry and the functions that each performs.

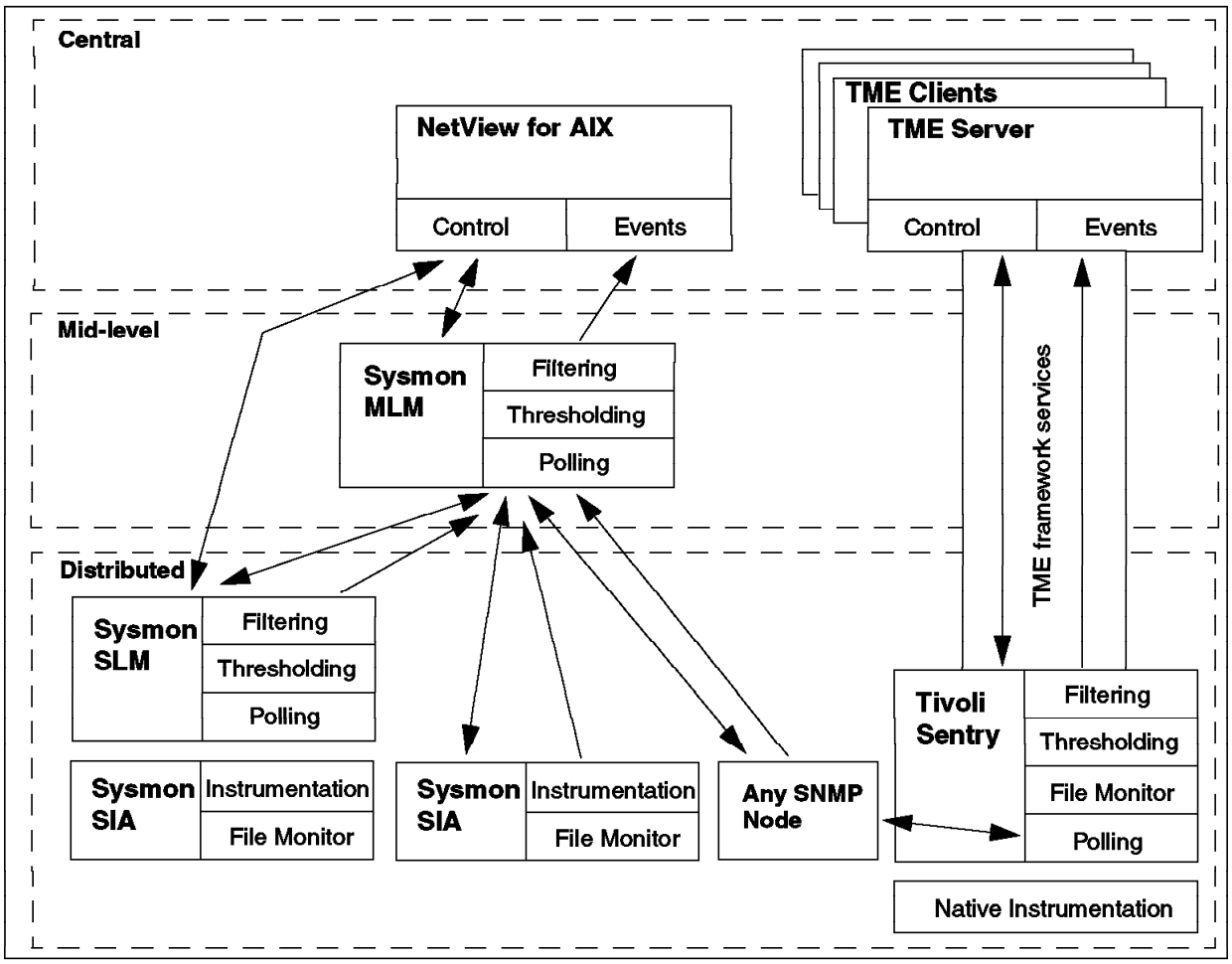



*Figure 1. Systems Monitor and Sentry Components and Functions*

### 1.1.1 Distributed Monitoring Now

Following the merger of Tivoli into IBM, in March 1996, the TME 10 family of products was born. This is based on the pre-existing TME architecture, with selected SystemView products imported into it. The TME 10 roadmap sets a number of ambitious objectives, staged over a two-year timescale. The initial phase integrated the marketing of the two systems management product lines, by aligning terms and conditions and axing products that are no longer strategic. The second phase seeks to integrate the remaining products using the TME framework as a base.

Under the first phase of the roadmap, TME 10 Distributed Monitoring is a bundling of Tivoli/Sentry with Systems Monitor for AIX.

### 1.1.2 Into the Future

Tivoli Systems' objective is to deliver consistent, feature-rich, multi-platform solutions based on the Tivoli framework. After technical assessment of the two distributed monitoring product lines, Tivoli has decided on the following strategy for the second phase:

1. TME 10 Distributed Monitoring is simplified to contain only the Tivoli/Sentry function. This is the prime application for monitoring of distributed systems.

2. Systems Monitor Mid-Level Manager (MLM) is retained and becomes a component of TME 10 NetView for AIX. MLM is designed to perform polling and network discovery on behalf of NetView. This decision affirms its role as the key to improving the scalability of NetView.
3. Systems Monitor System Level Manager (SLM) and System Information Agent (SIA) are withdrawn. SIA will be available for download from the World Wide Web for several months to assist migration.

The objective of this book is to describe how to migrate from Systems Monitor to TME 10 Distributed Monitoring. We describe a methodology for planning and implementing the migration process and show examples of using TME 10 Distributed Monitoring in various configurations. But first we describe how the two products work.

# Chapter 2. Systems Monitor in Detail

In this chapter we describe the detailed operations of Systems Monitor. If you are familiar with the product you may wish to skip this section. If, however, you do not understand the detailed operation of it we recommend studying this section, because otherwise the migration details in Chapter 4, “Migration Methodology and Tools” on page 25 will not make sense.

## 2.1 SNMP

We already mentioned in 1.1, “Distributed Monitoring History” on page 1, that Systems Monitor uses SNMP as its management protocol. The SNMP protocol defines a manager/agent relationship and a group of commands to retrieve information from agents running on managed nodes.

SNMP-based systems management uses a client/server approach.

- The manager (client)

  A manager acts as a client application that controls its management domain, usually nodes within a TCP/IP-based network, and performs monitoring tasks.Typical SNMP managers are TME 10 NetView and TME 10 NetView Mid Level Manager (formerly IBM Systems Monitor MLM). Managers control the network by polling remote nodes for system information and status. In addition a manager is capable of receiving asynchronous events called *traps*. Based on the polled or received information, the manager application executes defined actions such as operator notification or restart actions.

- The agent (server)

  An agent provides information to the Manager. It is responsible for recording and maintaining system data that can be retrieved by the manager using the polling mechanism described above. In addition, the agent may warn of critical conditions by sending traps to the manager application.

Although it is common to refer to the whole management process as SNMP, in fact SNMP refers only to the protocol used to communicate between agent and manager. SNMP stands for *Simple Network Management Protocol*. SNMP defines the relation between agent and manager and provides the commands to get information from the agent and set certain data.

SNMP polling requests reference an abstract database, called the *Management Information Base*, or MIB. The MIB defines a hierarchy of managed objects. When the SNMP manager polls for (using an SNMP GET request) the value of a MIB object, the SNMP agent is responsible for returning the value of it as a response. In fact, the value that is returned is of an *instance* of the object. An object can have multiple instances, thereby allowing tabular information to be maintained. Updates are handled in the same fashion; the manager specifies the new value of a MIB instance in an SNMP SET request and the agent is responsible for updating the managed system.

Originally, SNMP was introduced to perform network management. Thus, the basic MIB provided with most SNMP agents concentrates on network-related information and the data that is useful for systems management purposes is very limited. Fortunately, the MIB structure permits you to extend its scope without

limit, so it is possible to create an SNMP agent that will handle any type of information you choose. Sometimes this is achieved by rewriting the agent code, but normally an SNMP agent implements a *subagent interface*. This is an API that allows extension code to be plugged into the base agent to provide extended MIB support. There are several different subagent interface flavors. The most common are the Distributed Programming Interface (DPI) and the SNMP Multiplexor (SMUX).

## 2.2 Systems Management with NetView for AIX and Systems Monitor

To allow systems management based on SNMP, various elements are involved. Figure 2 shows the flow between the different parts of a distributed management environment using IBM Systems Monitor MLM and SIA and TME 10 NetView.

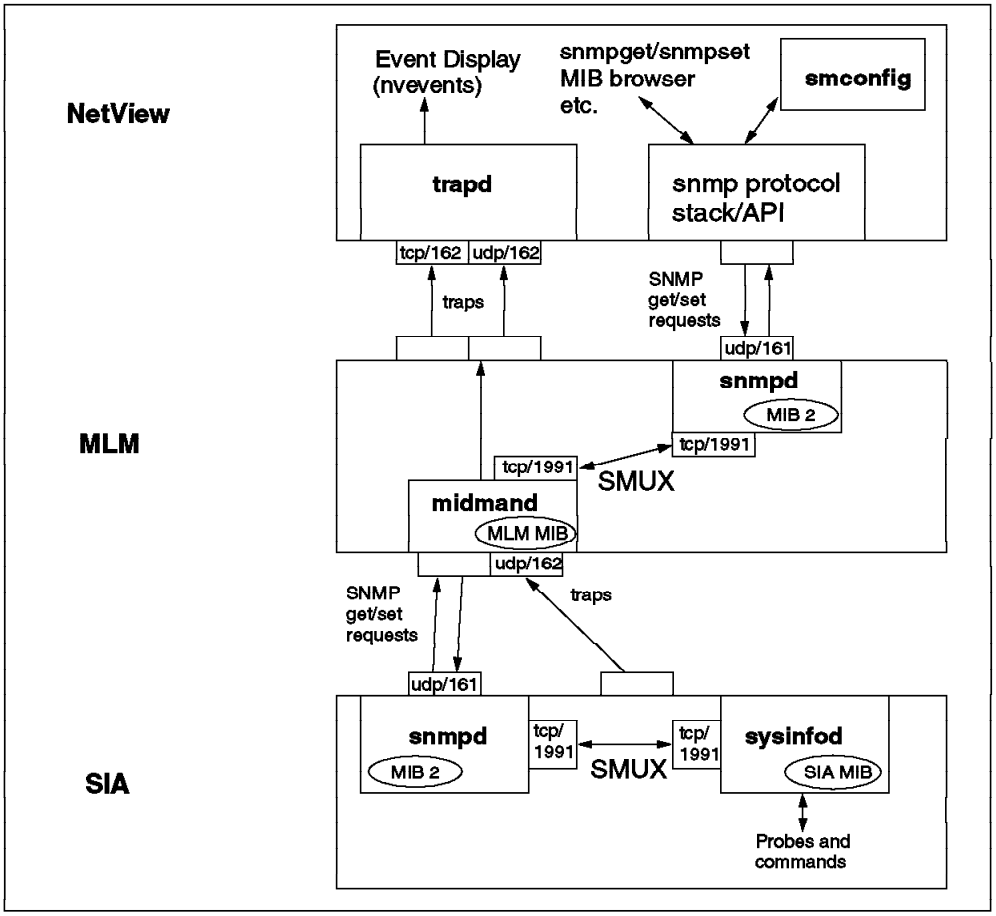



*Figure 2. Connections between Systems Monitor Components*

The elements of a Systems Monitor distributed monitoring environment are:

- NetView for AIX
- Systems Monitor MLM
- Systems Monitor SIA
- Systems Monitor SLM
- Systems Monitor Configuration Application (smconfig)

### 2.2.1 TME 10 NetView

NetView for AIX (now called TME 10 NetView) is the SNMP Manager and focal point for distributed networks. It provides a graphical end user interface. NetView is capable of discovering new nodes in a network and produces graphical representations of the network, called maps. You can perform limited systems management tasks with NetView for AIX. NetView for AIX also provides the base function for the Systems Monitor graphical user interface, smconfig, which allows you to configure MLM, SLM and SIA agents from a central point.

### 2.2.2 System Information Agent (SIA)

The System Information Agent (SIA) is an SNMP SMUX Subagent. The SIA extends the standard MIB database by adding large numbers of system-related variables. These variables are dependent on the system where the SIA executes. System Information Agents are available for the following platforms:

- AIX
- OS/2
- SunOS
- SUN-Solaris
- HP-UX
- NCR UNIX

The SIA MIB differs in the quantity of information it can provide depending on the platform it is running on. In addition, SIA can send traps to any SNMP manager or to the Systems Monitor SLM or MLM. Note that the SIA does not, itself, have any management capability, in the sense of querying the MIB for thresholding and data collection. Systems Monitor provides the SLM and MLM to perform these functions in a distributed way.

#### 2.2.2.1 SIA MIB Field Meanings

The SIA has a large extended MIB. For effective system management, we have to select the most useful objects from it. The following table describes the different groups within the SIA MIB and summarizes their functions. The command column shows the command that could be issued locally to find the information provided by the MIB variable.

*Table 1. SIA MIB Tables*

| Group | | Contents | Example | Command | MIB |
|---|---|---|---|---|---|
| System Description | | The system information of the SIA machine. | Hostname, OS, Version, cpu_id, etc. | uname, etc. | smSiaSystemDescription |
| System Configuration | | Configuration information of the operating system on the SIA machine. | .A number of procs, Page size, File table size, etc. | lsattr -E -l sys0, etc. | smSiaSystemConfiguration. .. |
| System Device | List | Device information about the SIA machine. | Installed device name, desc, VPD, attribute and location, etc. | lsdev -C, lscfg -v, etc. | smSiaSystemDeviceList... |
| | Token-Ring | Token-ring device and detailed performance information of the SIA machine. | Token-Ring device name, Attribute, VPD, RDTO, MAC addr, Detailed performance data, etc. | lsattr -E -l tok0, netstat, etc. | smSiaSystemDeviceToken Ring... |
| | Ethernet | Ethernet device and detailed performance information of the SIA machine. | Ethernet device name, Attribute, VPD, RDTO, MAC addr, Collision, Performance data, etc. | lsattr -E -l ent0, netstat, etc. | smSiaSystemDeviceEthern et... |
| | X.25 | X.25 device, configuration and detailed performance information of the SIA machine. | X.25 configuration information and Performance data, etc. | lsattr, etc. | smSiaSystemDeviceX25... |
| System Paging Information | Free Paging Space | Information about free paging space on the SIA machine. | Free paging space, etc. | svmon, vmstat lsps, etc. | smSiaSystemFreePagingSp ace... |
| | Paging Space | Attributes of the paging space. | vg, pv name of the paging space, etc. | lsps etc. | smSiaSystemPagingSpace. .. |
| | Paging Statistics | Various paging statistics about the SIA machine. | Count of pagein, Pageout and Pagefault, etc. | vmstat etc. | smSiaSystemPagingStatisti cs... |
| System File System | | File system information about the SIA machine. | Filesystem name, Size, Utilization, Inode count and Mount point name, etc. | lsfs, df, mount, lsvg, lspv etc. | smSiaSystemFileSystem... |
| System Subsystem | | AIX subsystem details of the SIA machine. | Subsystem name, PID, Status and Subsystem group name | lssrc, etc. | smSiaSystemSubSystems... |
| System Process | | Various data about running processes. | Process name, PID, GID, CPU time (user/system), Pagefault, Priority, Status, VM size, Memory utilization, CPU utilization and Start time, etc. | ps, etc. | smSiaSystemProcess... |
| System Users | | Information about logged-in users. | User name, Login time and PID, etc. | who, ps, finger, w, etc. | smSiaSystemUsers... |
| System Utilization | CPU | Information about CPU utilization on the SIA machine. | Percent CPU in user mode, System mode, Idle mode and Wait mode, etc. | iostat, vmstat, etc. | smSiaSystemUtilizationCPU ... |
| | Kernel | Information about the kernel status of the SIA machine. | The number of context switch, System call, System read, System write, Fork and Exec, etc. | iostat, vmstat, etc. | smSiaSystemUtilizationKern el... |
| | Iostat | Information about I/O status of the SIA machine. | Transfer rate, Volume, Read volume and Write volume, etc. | iostat, etc. | smSiaSystemUtilizationIost at... |
| System Miscellaneous | | Miscellaneous information about the SIA machine. | System time, unallocated space in vg, etc. | date, lsvg, etc. | smSiaSystemMiscellaneous ... |

#### 2.2.2.2 SIA File Monitor and Command Tables

Two other important functions of SIA are the File Monitor table and the Command table. The File Monitor table allows you to monitor the characteristics of a file for changes and also to monitor the data in a file searching for specific strings. The Command table allows you to extend the data provided by the SIA by executing any line command. The output from the command is returned as the response to an SNMP get request.

### 2.2.3 Mid Level Manager (MLM)

The TME 10 NetView Mid Level Manager (formerly IBM System Monitor Mid Level Manager) is a distributed Network Manager. It collects information and receives traps from any other SNMP agent (not only SIAs). It provides thresholding and filtering functions so that you can use MLM to poll for problems and then send them as SNMP traps to a central SNMP manager, or take automated action. Other significant capabilities of the MLM are its polling and discovery features. The MLM can poll a group of nodes for status purposes and discover new nodes in its subnet. This reduces the load on TME 10 NetView and allows a true distributed management of SNMP nodes.

Although Figure 2 on page 6 shows the monitored node as a Systems Monitor SIA agent, in fact any node with a standard or extended SNMP agent can be monitored using MLM.

### 2.2.4 System Level Monitor (SLM)

The System Level Monitor (SLM) runs on a node where the SIA is installed and offers thresholding, analysis and filtering of information for that local node. Functionally, the SLM is identical to the MLM, except for the following restrictions:

1. Node discovery is not supported.
2. Status monitoring of remote nodes is not supported.
3. Thresholding is only performed on the local node.
4. Trap reception from another node is not possible.

### 2.2.5 Systems Monitor's Use of SNMP

As you can see, all components of Systems Monitor use SNMP to exchange information:

- Both NetView and MLM use SNMP get requests to poll MIB variables from their managed agents.
- NetView and the GUI component of Systems Monitor, smconfig, use SNMP set requests to configure and control the MLM and SLM. SNMP set requests are also used to configure the Command table and File Monitor table of the SIA.

SNMP agents, which are required on both manager and agent nodes, use traps to send unsolicited information to their assigned manager. In case of an MLM being the manager, these traps can be forwarded to an upper-level manager such as NetView or used to trigger systems management actions on behalf of the MLM. This allows, in addition to the MLMs polling capabilities, some offload of management tasks to a local network segment, reducing the required bandwidth.

Systems Monitor applications offer various ways to access, display and process the collected data. We will discuss the different ways to work in an SNMP-based distributed environment.

## 2.3 Using TME 10 NetView and Systems Monitor Tools

There are a number of tools for monitoring and controlling a Systems Monitor configuration:

**MIB Browser** A MIB Browser lets you browse and set MIB variables on a target node. You can enter the address of a node you want to examine directly into the browser dialog or you may select a node in one of NetView's graphical network representations (maps). TME 10 NetView comes with an integrated MIB Browser, giving you a dialog like the one in Figure 3, which can be used in conjunction with several other features of TME 10 NetView.

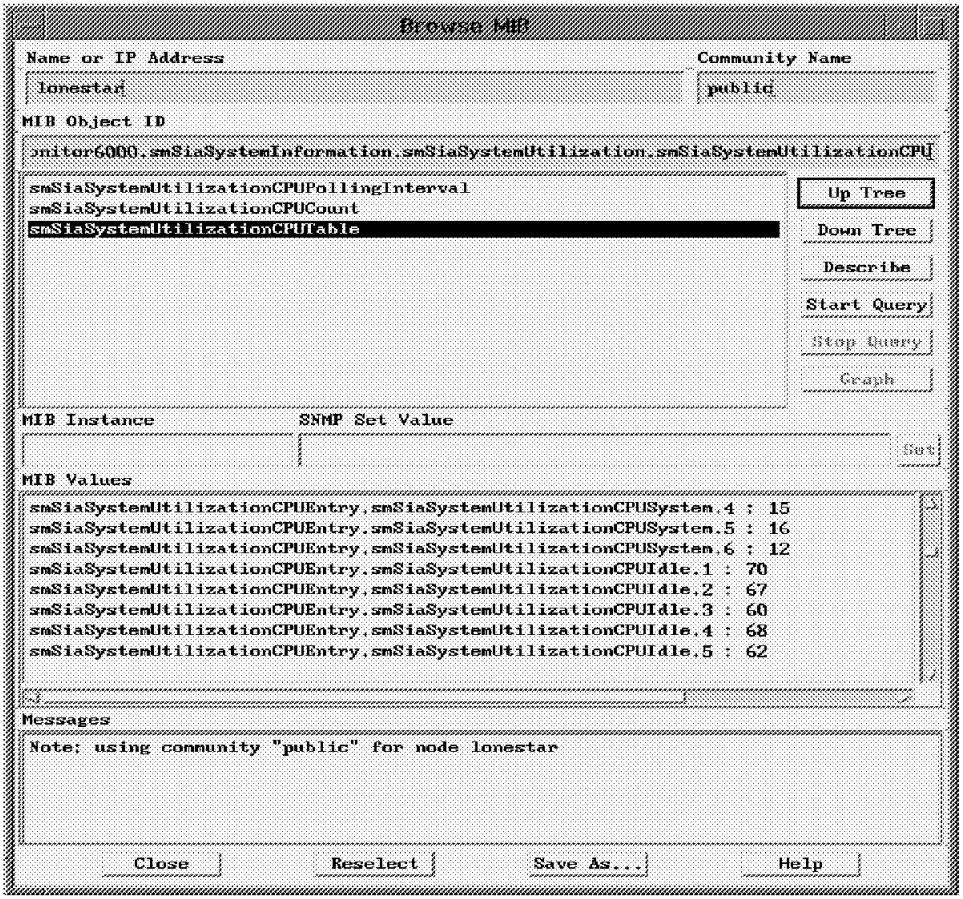



*Figure 3. The NetView MIB Browser*

**Control Desktop** The control desktop is the final sink for traps forwarded to TME 10 NetView. By default, all traps received by TME 10 NetView that are not directly intercepted by other TME 10 NetView applications will be directed to the control desktop. The control desktop offers various filter capabilities. In addition you may launch more than one

desktop and direct filtered trap information to those desktops.

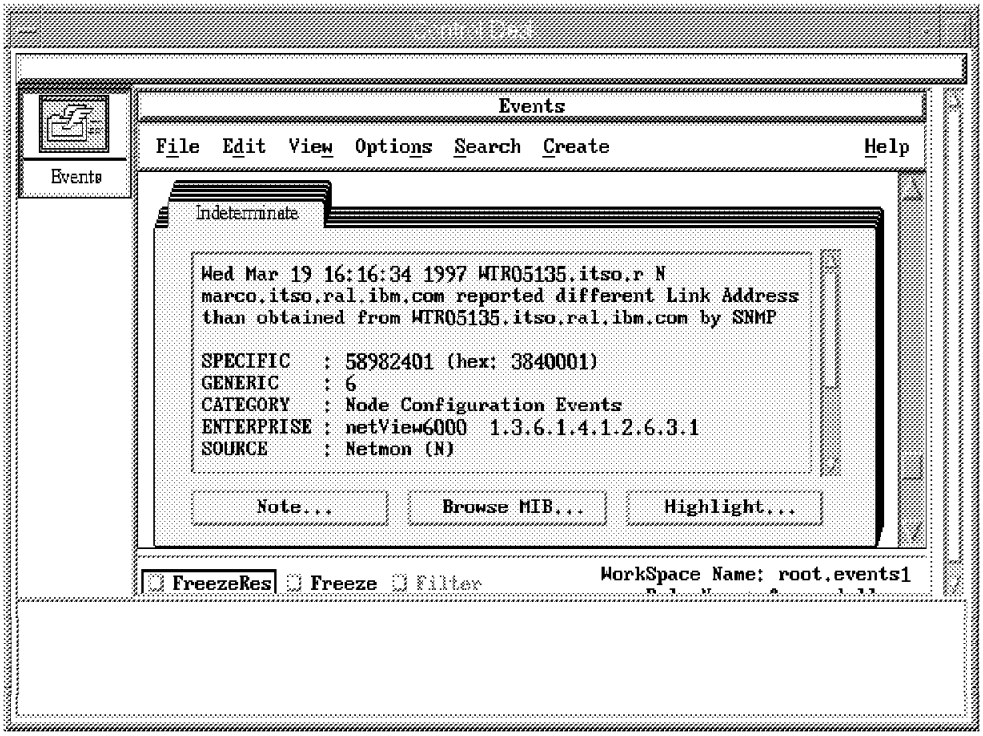



*Figure 4. The NetView Control Desk*

**NetView Maps**

The main TME 10 NetView tool you will use is its various graphical representations of the network. NetView displays the entire network it discovers in a group of maps. These maps show the status of the nodes in your network and are updated dynamically. TME 10 NetView inserts newly discovered nodes into the correct maps. TME 10 NetView then display the node symbol in different colors according to the actual status of the node, as determined by regular polling.

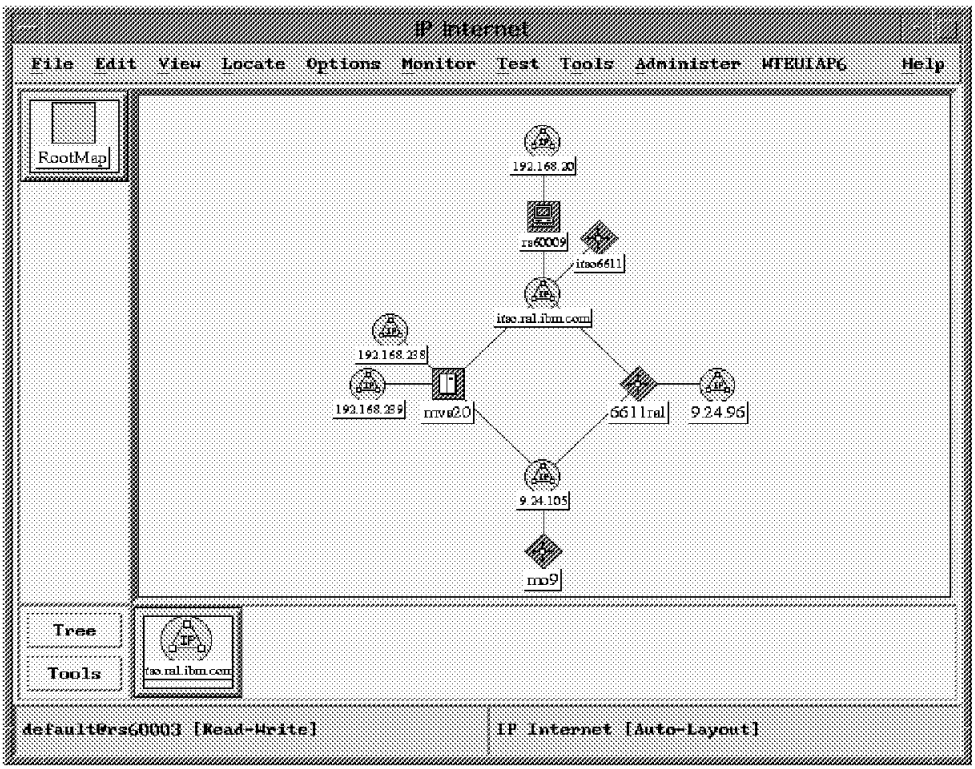



*Figure 5. A Typical TME 10 NetView Map*

**APM**

APM, short for Agent Policy Manager, is another useful application of TME 10 NetView. You can customize TME 10 NetView to automatically detect MLMs in your network and distribute managment duties automatically to those MLMs. If you decide to do so, NetView will offload discovery and status polling to discovered MLM nodes. You can offload status polling, discovery or both. TME 10 NetView then receives every change in the subnet controlled by an MLM via traps. In a distributed environment, where MLMs control subnets, APM helps you to distribute monitors across the network. APM provides you with dialogs to define SIA file monitors and MLM/SLM threshold definitions for groups of nodes. These groups are actually dynamic node collections. You define the collections by specifying the characteristics that nodes in the group share. APM will apply the monitors you define to the current group of nodes in the collection and will update all of them whenever you change the monitor definitions. It will also apply the current monitor set to any nodes that are dynamically added to the node collection. Figure 6 on page 13 shows a typical APM configuration screen, defining a file monitor on a collection of SIA nodes.

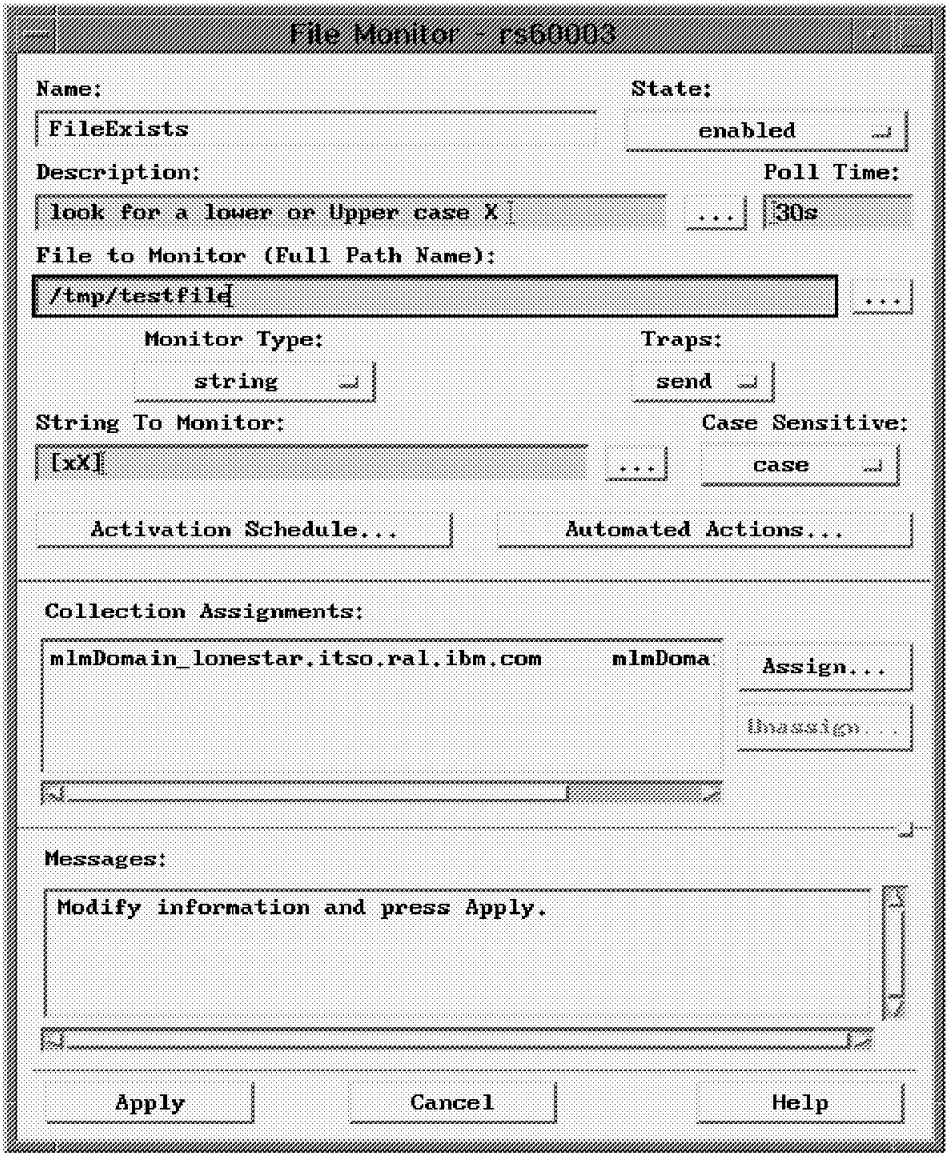



*Figure 6. An APM File Monitor Definition*

**smconfig**

The configuration details as well as the thresholds and monitors you define are held in a configuration file on the target node where Systems Monitor components reside. APM provides one way to update the configuration from a central point. Systems Monitor also provides a graphical user interface, the Systems Monitor Configuration Application (see Figure 7 on page 14).

.

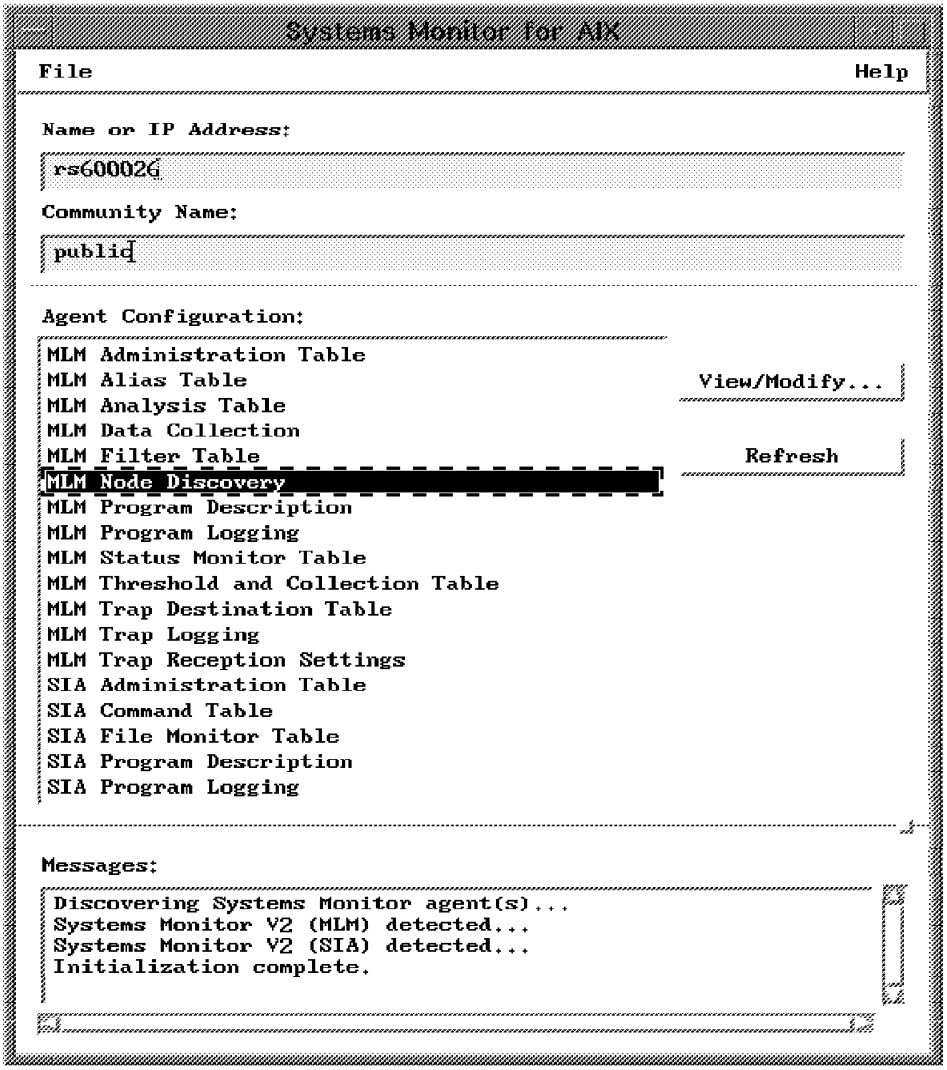



*Figure 7. TME 10 Systems Monitor Configuration Application*

This GUI is the easiest way to update the configuration of individual Systems Monitor agent nodes. The configuration information itself is organized into MIB tables and the GUI provides separate dialogs for each of the tables.

This has been a very brief introduction to the functions and components of Systems Monitor. For more details about the product and examples of its use, refer to *IBM Systems Monitor Anatomy of a Smart Agent,* SG24-4398.

# Chapter 3. TME 10 Distributed Monitoring in Detail

In this chapter we describe the detailed operations of TME 10 Distributed Monitoring (previously called Tivoli/Sentry). If you are familiar with the product you may wish to skip this section. If, however, you do not understand the detailed operation of it we recommend studying this section, because otherwise the migration details in Chapter 4, “Migration Methodology and Tools” on page 25 will not make sense.

TME 10 Distributed Monitoring 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 fuller 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.

## 3.1 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 8 shows a schematic view of TME.

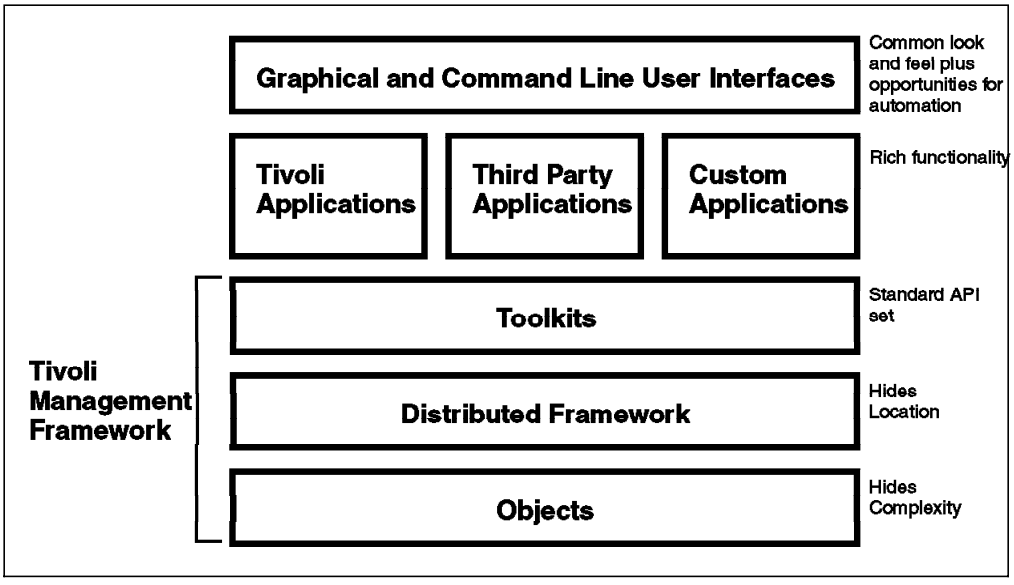



*Figure 8. TME Architecture*

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.

### 3.1.1 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. Why, then, is one designated as 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 9 on page 17 shows how TMRs are put together.

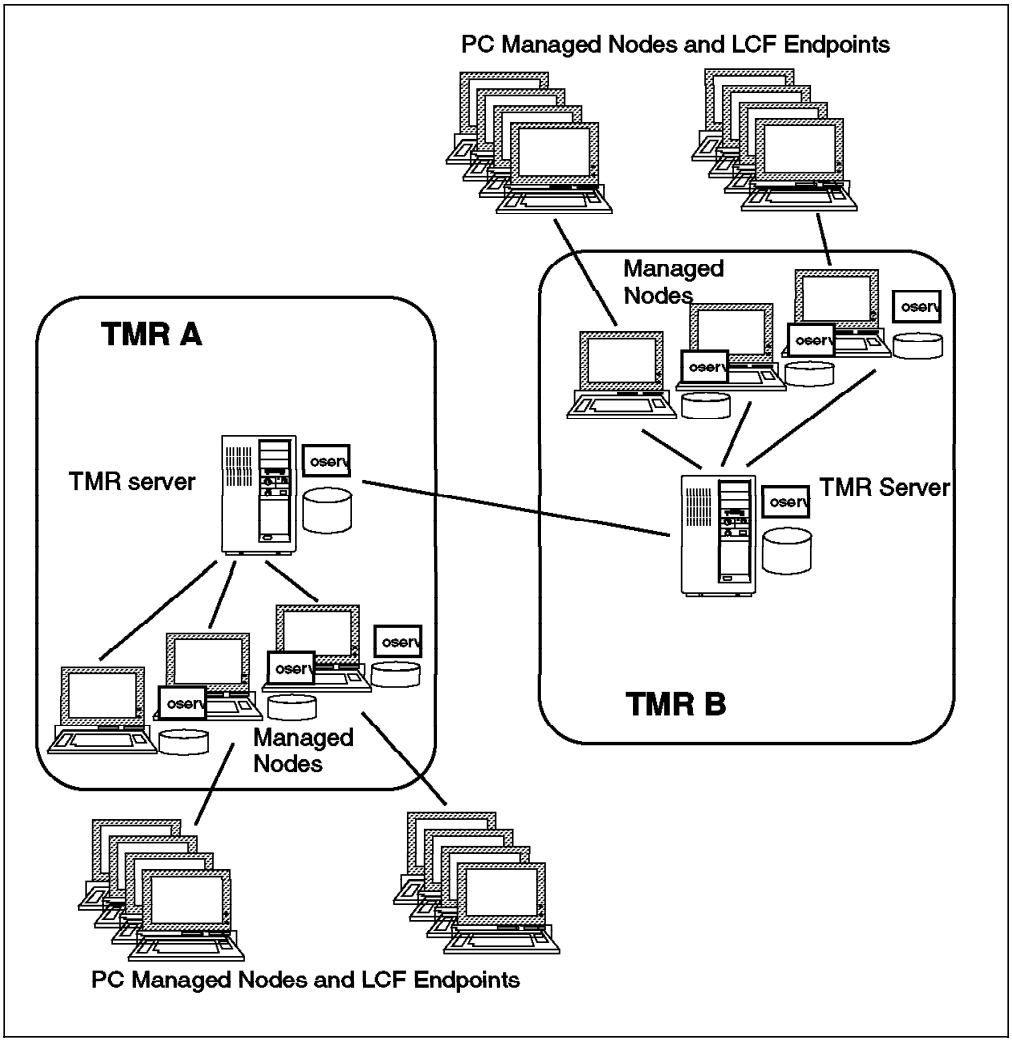



*Figure 9. TMR Components*

## 3.1.2 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, such as 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.

### 3.1.3 Management by Subscription

With Systems Monitor it is possible to modify the monitors for an individual system, so that all systems can be different. It is also possible to use APM to set a monitoring policy and apply it to a group of nodes (see 2.3, “Using TME 10 NetView and Systems Monitor Tools” on page 10). By contrast, in TME, *all* applications are configured based on policy.

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 depend 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 10 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 analyses, 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.

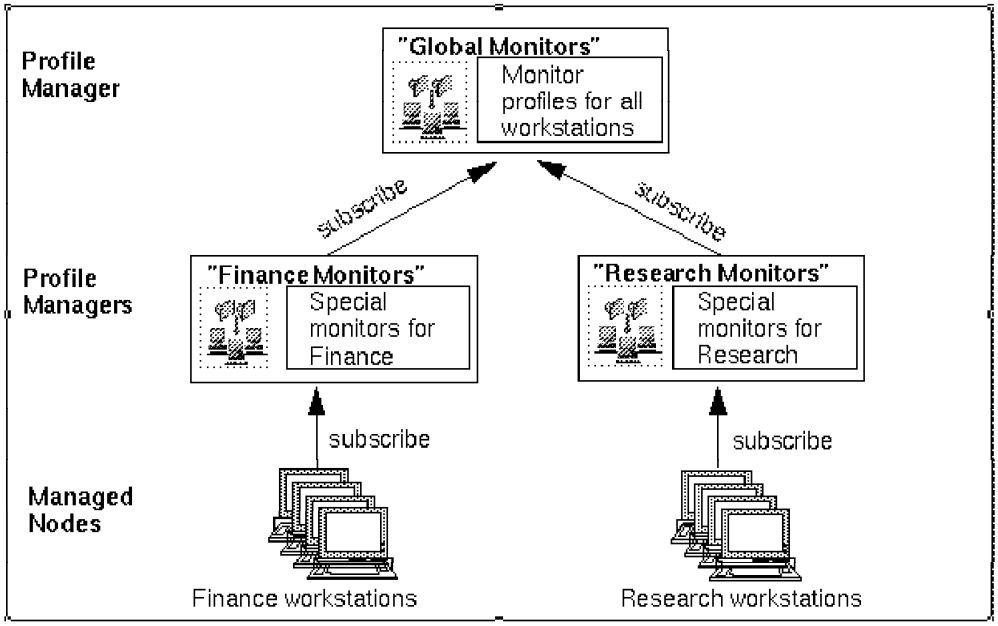



*Figure 10. Hierarchy of Profile Managers*

## 3.2 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* triggers 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, such as a program or a TME task.

#### 3.2.1.1 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 systems 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.

## 3.3 Under the Covers of TME 10 Distributed Monitoring

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

1. Creating a monitor
2. Executing a monitor

### 3.3.1 Creating a Monitor

Figure 11 shows how a monitor is instantiated on a managed node.

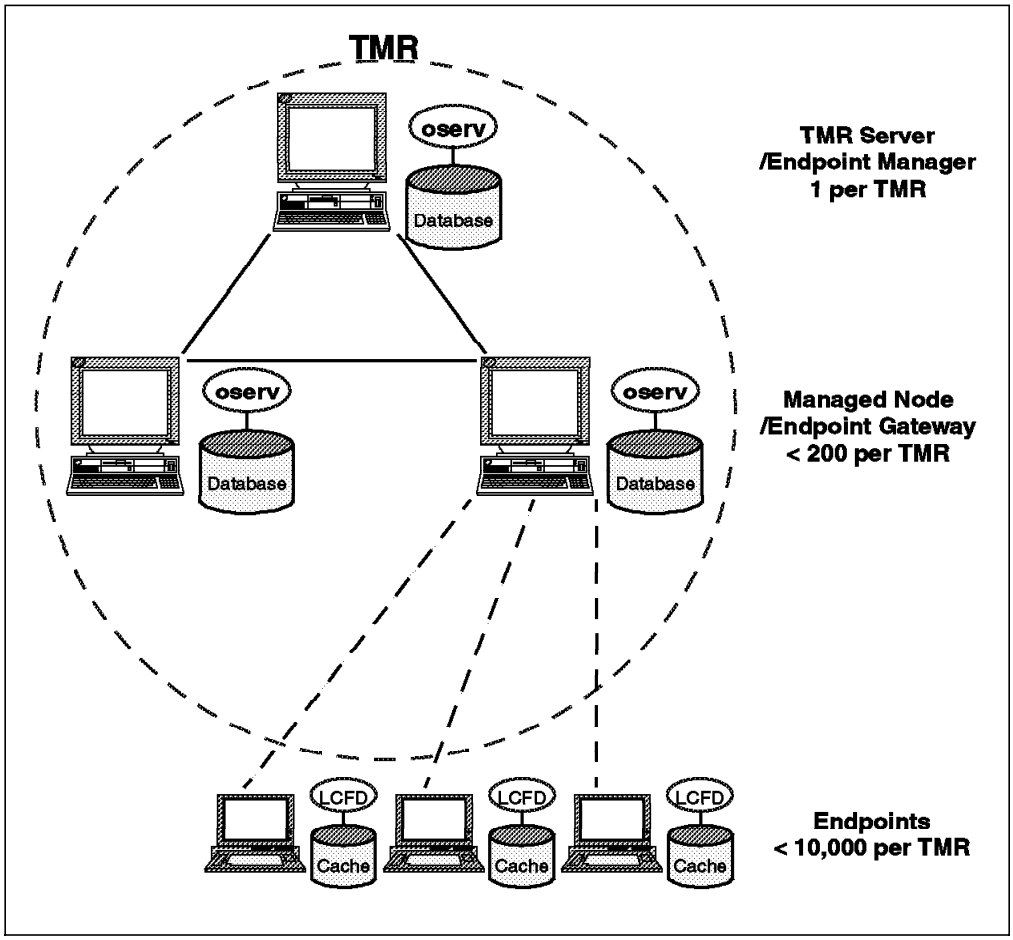



*Figure 11. Creating a Monitor*

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, which are effectively a package of code and data containing the instructions for the new monitor. Because this process 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.

#### 3.3.1.1 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.

We discuss how to detect and deal with this problem in Chapter 6, “Installation Notes and Trouble Shooting” on page 131.

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 12 shows what happens.

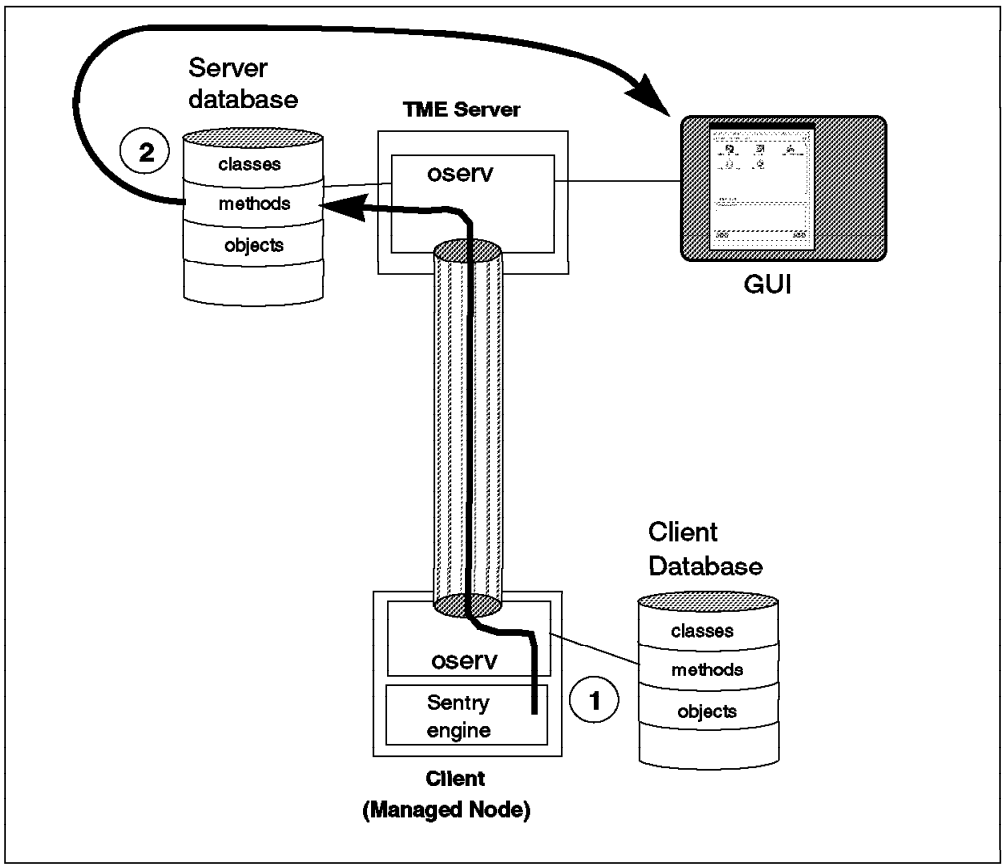



*Figure 12. Executing a Monitor Action*

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.

## 3.3.2 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 predefined 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. In this book we will show how to do this to send events to TME 10 NetView, for example.

### 3.3.2.1 Generating Pop-Up Messages

A pop-up message can be defined to appear on any TME administrator's desktop, as shown in Figure 13.

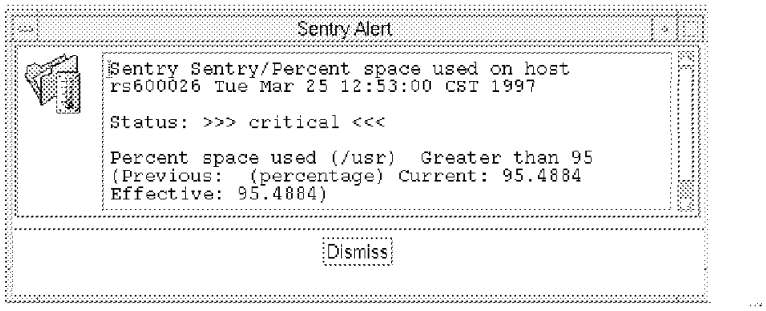



*Figure 13. An Event Pop-Up Message*

### 3.3.2.2 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 pop-up, although the Notices icon changes to show that unread notices have arrived (see Figure 14).

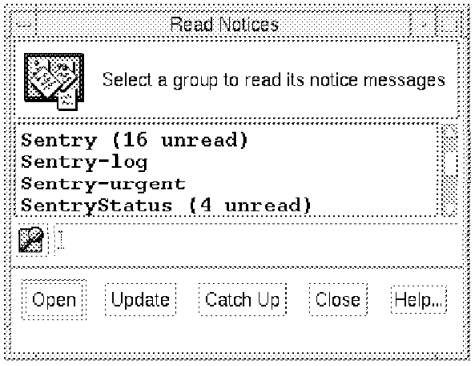



*Figure 14. Sentry Notices*

### 3.3.2.3 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 that allows events to be correlated regardless of generation time and source. These rules provide filtering of potentially high numbers of events (see Figure 15).

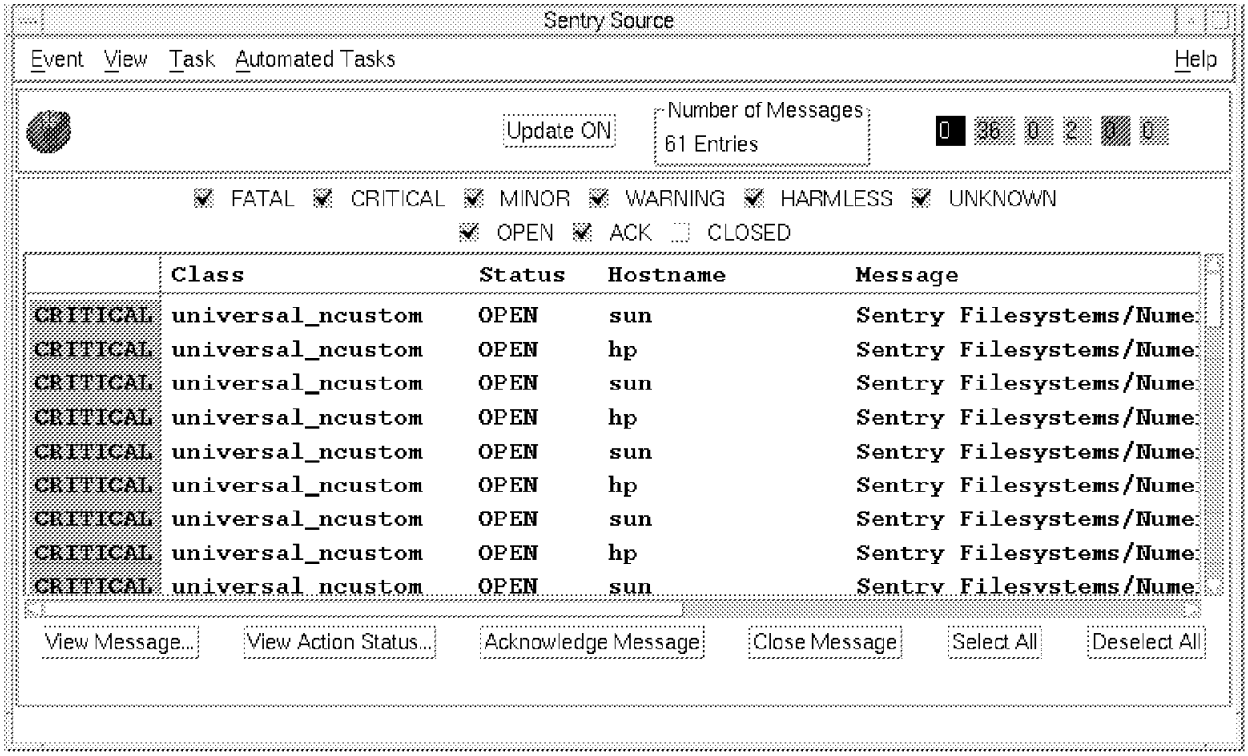



*Figure 15. Events Sent to the T/EC*

### 3.3.2.4 TME 10 NetView

TME 10 NetView provides an event handling process, which displays SNMP traps received from the network and internally generated NetView events. If you are migrating from a Systems Monitor environment, you already use TME 10 NetView for centralized event display. If you want to keep using NetView in this way, Sentry provides two techniques for passing events into NetView:

1. You can use a TME task to run the NetView event command directly on the system where NetView is running.
2. You can use an automatic command to send an SNMP trap to NetView.

We will show examples of how to do both of these in Chapter 5, “TME 10 Distributed Monitoring Examples” on page 43.

NetView runs on AIX, Sun Solaris and NT platforms. With all versions you can view events as they arrive in a window on the NetView system. With the UNIX version you can also use a Java-enabled Web browser to access NetView (V5 only). See Figure 16.

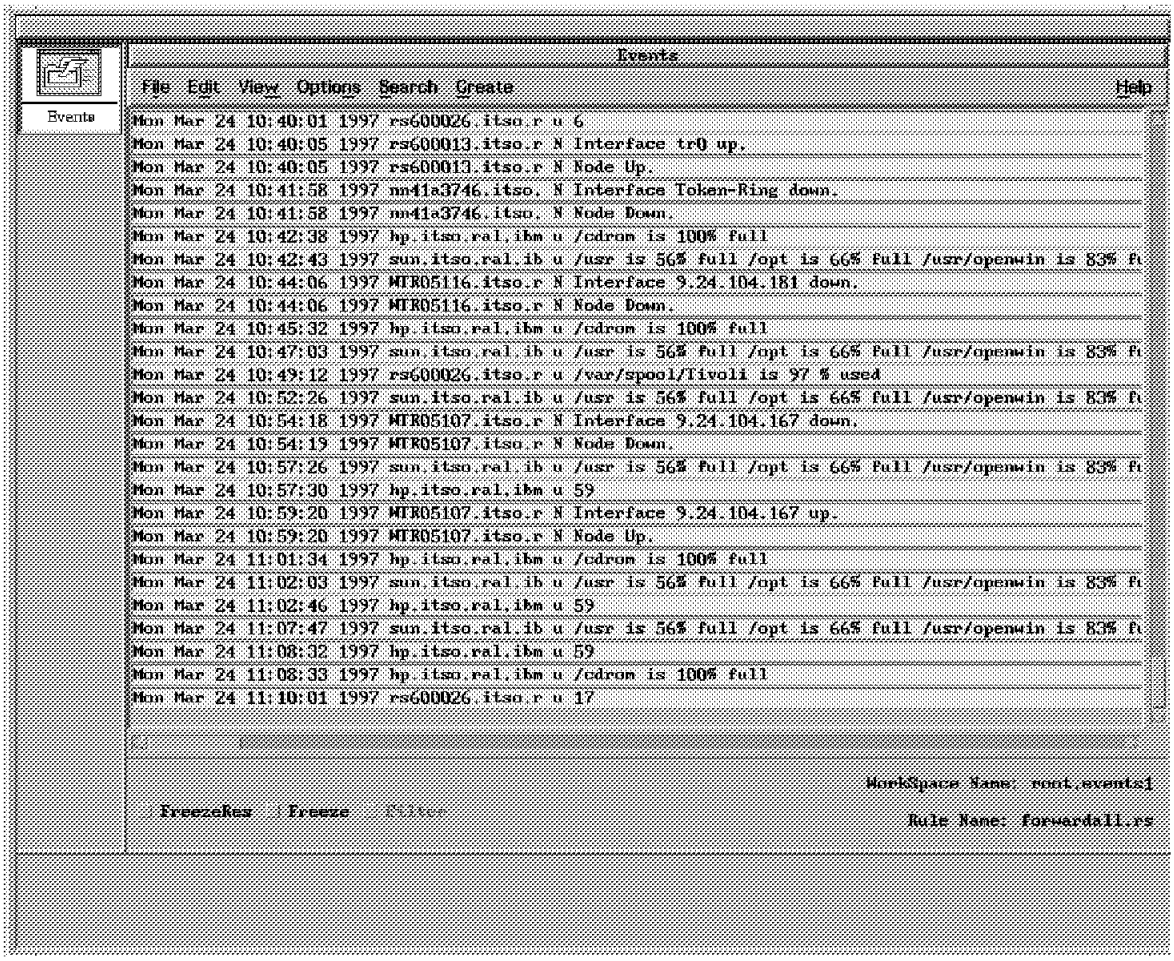



*Figure 16. Events Arriving at TME 10 NetView*

# Chapter 4. Migration Methodology and Tools

Now that we have an understanding of the differences between the old and new distributed monitoring environments, we are ready to think about a migration project. In this chapter we work through the steps involved in the migration and introduce some tools to help.

## 4.1 Pre-Migration Planning

Before you can go into the details of migrating your Systems Monitor environment, you need to plan the installation of the infrastructure needed by TME 10 Distributed Monitoring. Essentially, this means that the TME platform needs to be installed and configured. This involves planning the layout of your TMRs and positioning the TMR server(s). This need not be complex, but it is worth doing carefully, because any poor decision taken at this stage can cause extra work later.

The following list contains a number of points that you should consider:

- Number and placement of TMRs

  The general rule is that the fewer TMRs you have, the better. There is a limit of about 200 to the number of managed nodes that can be placed in any TMR. If all of the nodes are equally well connected to each other, you should aim to make the TMRs maximum sized - 200 nodes per server. If there are some slow speed links in the network, you should arrange connected TMRs on each end of the link. This is because the TMR servers implement the mdist (multiplexed distribution) repeater function over TMR-TMR links. In other words, any monitor profile that you are distributing to systems on the far side of the link will only be sent once to the TMR server and will then fan out.

  The introduction of LCF changes the arithmetic. You can have many more (thousands) of LCF endpoints connected to one TMR. Furthermore, each endpoint gateway acts as an mdist repeater, so slow speed links are more easily catered for. LCF support will be available in TME 10 Distributed Monitoring for most operating systems by the end of 1997.

- Selection of TMR servers

  Each TMR needs a single TMR server. In normal operation, the TMR server function is not a major consumer of system resource, but it *must* be available 100% of the time. You can certainly share the server platform with some other application if the opportunity is there. Once again, LCF changes the equation. More systems within a TMR translates to more work for the server to do, such as processing monitor actions and automation.

- Configuring Administrators

  Systems Monitor does not have a well-defined administrative structure. Any NetView user who has the ability to execute SNMP get and set requests can be an administrator. When you migrate to TME 10 Distributed Monitoring you could retain this simple structure, by defining a single administrator ID and giving the password to anyone that needs administrative access. However, a better approach is to exploit the administrative control that TME offers by defining different user IDs for everyone who needs one. You can

then use different policy regions to control who can do what to which part of the distributed system population.

- Event Handling

  You need to decide what your policy is for presenting event information to administrators. You may decide not to use a single mechanism. For example, it may make sense for some administrators to receive events from specific types of monitors via e-mail or pager, in addition to sending them to a consolidated event display.

## 4.2 The Overall Migration Picture

One thing that quickly emerges from a study of the migration options is that it is not a simple environment. The existing Systems Monitor environment will consist of a number of target nodes, on which the monitored resources reside. These systems could be a number of different types of operating systems, or they could be network devices such as routers and hubs. Monitoring of the resources may be done locally (using Systems Monitor SLM), or at a mid-level manager (Systems Monitor MLM) or centrally (NetView for AIX).

The target environment will include TME 10 Distributed Monitoring installed on some of the monitored nodes. However, it will also include resources that cannot be migrated directly (SNMP nodes, for example) and resources monitored by proxy. The mid-level component could therefore consist of TME 10 Distributed Monitoring, or Systems Monitor MLM, or both. Finally there are a number of different options for displaying and handling event information, as we discussed in 3.3.2, “Displaying Events” on page 22.

Figure 17 shows the post-migration possibilities and breaks down the managed domain into five different categories.

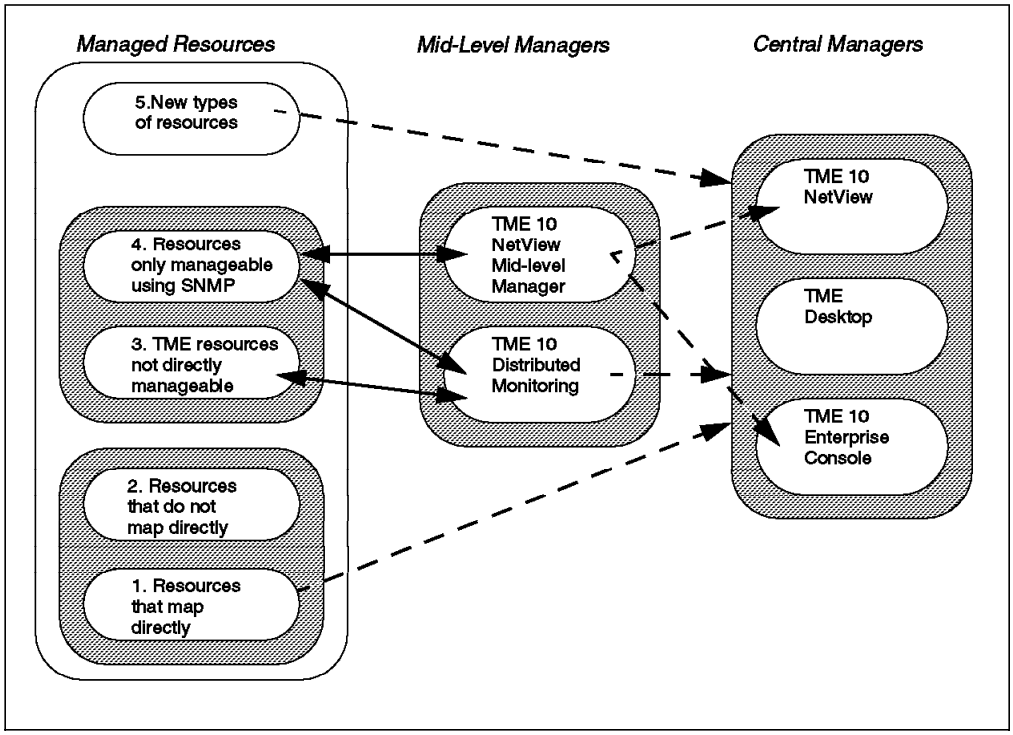



*Figure 17. Pieces of the Migration Puzzle*

The managed resource categories are as follows:

1. Resources that map directly. This means cases where there is a TME 10 Distributed Monitoring monitor that performs exactly the same function as the Systems Monitor monitor that it replaces. Migration of this category can be a semi-automatic function. We provide examples of migration scripts to do this.
2. Resources that do not map directly. This means that, although it is supported on the managed node, there is no TME 10 Distributed Monitoring monitor that does exactly what the old monitor did. In other words, a new monitor type has to be created, using shell scripts or programs, or alternatively a decision has to be made whether to continue monitoring in the same way.
3. TME resources not directly manageable. These are the cases where either the TME platform or Sentry has not been ported to the particular system type. It is possible to use proxy techniques to monitor these resources using TME 10 Distributed Monitoring.
4. Resources that are only manageable using SNMP. Obvious examples here are networking hardware, which support SNMP but do not have an operating system on which TME could be implemented. Another possible example is OS/2, which has an SIA available, but does not (at the time of writing) have Sentry support.
5. New types of resources. TME 10 Distributed Monitoring supports some operating systems that could previously only be effectively monitored by incorporating third-party products or SNMP MIBs, such as Windows NT and Novell NetWare.

Understanding how your current monitoring environment is divided among these categories is the key to the migration decisions you have to make, so we now describe a series of steps that will help you categorize your migration problem.

## 4.3 Analyzing the Migration

Before you start, we recommend that you collect together the configuration files of all of the Systems Monitor agents in your environment (or gather representative samples, if many of them are similar). Systems Monitor keeps its configuration files under directory /var/adm/smv2. Alternatively, you can display the configuration through the Systems Monitor configuration interface (smconfig), but that may not be very efficient if you have a large number of systems with a lot of variation between them.

This checklist will help you to gain an understanding of what your migration project will involve. After you have worked through the list, you can concentrate on the details of the more difficult migration challenges.

1. *Analyze your existing Systems Monitor profiles*

   The first step is to analyze the current Systems Monitor configurations on each machine with the SIA, SLM or MLM installed. If you have been very well organized in the past, it is possible that many of the configurations share common monitors. If so, your migration task is much easier and the Sentry monitors can perhaps be placed in a single profile.

   If the monitors vary due to platform or application release, they will need to be placed in separate profiles and profile managers.

We recommend mapping out where each set of monitors fits into your framework.

2. *Do you have any SIA file monitors?*

   The file monitoring capabilities of SIA map to equivalent Sentry functions as shown in the following table; in other words they mostly fall into migration category 1 as shown in Figure 17 on page 26. Check the configurations of all of your SIA nodes for this type of file monitor. In general, you can migrate them using the techniques described in 4.5, “Practical Migration Techniques” on page 31.

   The table shows the SIA file monitor option and the equivalent TME 10 Distributed Monitoring monitor. Generally speaking, the SIA version has fewer trigger options, and the comments describe which Sentry options to use to most closely replicate the SIA function.

*Table 2. SIA File Monitor Mapping*

| SIA File Monitor Type (smSiaFileMonitorType field value) | Equivalent Sentry Monitor | Comments |
|---|---|---|
| dataChange | File checksum | This Sentry monitor is in both the Universal and Unix_Sentry collections. It indicates whether the data in a file has been changed in any way. In SIA, you can only specify a monitor that looks for *any* change. In Sentry you have a number of options. To trigger a response on any change, as in the SIA case, you should set the **trigger when:** field to **Changes by** and the value to **1**.<br>To add this monitor using waddmon from the command line, use a monitor name of **filechk** and a relational expression of: **"+ >=" 1**. |
| statusChange | Check file permissions | This monitor is in both the Universal and Unix_Sentry collections. The SIA version checks file permissions and ownership, but Sentry only checks permissions.<br>There is also a difference in the response. Systems Monitor will generate an alert whenever the permissions are changed, because the test is *have the permissions changed within the monitoring interval?* The closest Sentry test is **Changes from**. This triggers an event when the permissions change from some predefined value, but does not flag subsequent changes. Specify the permissions as a string, for example **-rw-r--r--**.<br>To add this monitor using waddmon from the command line, use a monitor name of **fileperm** and a relational expression of: **"-<" "-rw-r--r--"**. |
| string | File pattern matches (Universal) or Occurrences in file (Unix_Sentry) | This monitor is in both the Universal and Unix_Sentry collections, although it has a different name in each. The SIA string monitor checks to see if a string or pattern has appeared in the specified file during the monitoring interval, and generates an event if so. The Sentry version performs checks on the total number of occurrences of a pattern within the file.<br>To configure an equivalent Sentry test to the SIA behavior you should set the **trigger when:** field to **Changes by** and the value to **1**.<br>To add this monitor using waddmon from the command line, use a monitor name of **countstr** and a relational expression of **"+ >=" 1**.<br>There is one further difference between the SIA and Sentry monitors. The SIA version allows you to use extended regular expressions for pattern matching (that is egrep style patterns). The Sentry monitors use basic regular expressions (grep style). In most cases this will not cause a problem, but if you need to use the extended form there is a monitoring collection in the sample package for this book which does it. Refer to “Dynamically Named Log Files” on page 99 for details. |
| exist, notexist | File checksum | This monitor checks to see if a file exists or not. The Sentry File checksum monitor returns a value of -1 if a file does not exist, so you can replicate the function by setting the **trigger when:** field to **Equal to** or **Not equal to** and the value to **-1**. |

3. *Do you have special case SIA file monitors?*

   There are some SIA file monitoring options that are not available in TME 10 Distributed Monitoring. In particular:

   - Execution of a command before the test is performed. We show an example in 5.4.2, “Dynamically Named Log Files” on page 99 of a way to handle this limitation.
   - Monitoring for file ownership changes. The SIA statusChange monitor checks for file ownership as well as permissions, but the matching Sentry monitor only checks for file permissions. We provide a sample monitor that provides the file ownership test capability. The file ownership sample is part of the package of sample code from the project; see Appendix B, “How to Get the Samples in This Book” on page 141. We describe how to create a monitoring collection in “Monitoring CPU Utilization” on page 105.
   - SIA provides some combination monitors that check for specific strings *and* changes to data or status. You are unlikely to meet one of these in a real configuration, but if you do you may have to replace it with multiple Sentry monitors.

4. *Do you have monitors of non-migrateable SIA resources?*

   Check the configurations of any MLMs and SLMs and also check the NetView for AIX data collection and thresholding facility for any references to SIA MIB variables. Check them against the list of SIA MIB variables shown in Appendix A, “Mapping SIA MIB Objects to Sentry Monitoring Collections” on page 137 . Build two lists:

   a. A list of monitors that do not map to a Sentry monitor. These are category 2 monitors, according to our designation.
   b. A list of the remaining category 1 monitors that can be migrated.

   We provide a shell script, chk_mig_cat, to help you do this. It will process MLM and SLM configuration files and report the migration category of any SIA monitors it finds. See Appendix B, “How to Get the Samples in This Book” on page 141 for details of how to get the shell script.

   The monitors in the latter list can be migrated using the techniques described in 4.5, “Practical Migration Techniques” on page 31.

5. *If you have category 2 monitors, decide what you will do with them.*

   Your only options for monitors that are not migrateable are:

   a. Re-create them using Sentry String script or Numeric script monitors.
   b. Remove them.
   c. Retain SIA on the particular systems where they are required until a better solution is available.

   We show a number of examples of script monitors in this book, which may help you estimate how much work is involved in the first option.

6. *Decide whether to monitor SNMP resources using MLM or Sentry.*

   If you ran the chk_mig_cat script it will have reported any non-SIA MLM monitors that you may be using. According to our designation, these are category 4 monitors. You can either choose to continue to monitor them with MLM, or to monitor them using the UserSNMP or rfc1213 monitoring collections of TME 10 Distributed Monitoring.

The main advantage of staying with MLM is that it is less work for you. MLM also has the capability to monitor all instances of a given MIB object using a wildcard definition. This can sometimes be a useful feature. On the other hand, using TME 10 Distributed Monitoring has the advantage of consistency with the rest of your monitoring process.

## 4.4 Systems Monitor and Sentry Can Co-exist!

Don't forget that, during the migration, Systems Monitor and Sentry can co-exist on the same machine. You may wish to have both types of monitoring running in parallel in order to ensure that no functionality is lost.

## 4.5 Practical Migration Techniques

For monitors that fall into the first category, that is, where there is a mapping between the old and new product function, there are two approaches you can take:

- GUI-to-GUI migration. This means opening the user interfaces of Systems Monitor and TME 10 Distributed Monitoring and creating migrated profiles manually. This is effective for a small configuration, but it would be laborious for anything bigger.
- Semi-automatic migration script. Sentry monitors can be created using the waddmon command, so it is possible to build a monitoring profile from a shell script. We provide a sample script, described in 4.6, "Semi-Automated Profile Migration" on page 35 which will input SIA or MLM configuration files and generate a stream of waddmon commands.

Whichever approach you take, there is no simple one-to-one mapping between functions. You will have to adjust your Sentry profiles and monitoring scripts to take account of the variations. The comments in Table 6 on page 137 will help you decide what adjustments are needed.

### 4.5.1 Migration of a Small Configuration

If you have a small number of SIA, SLM and MLM agents, then a GUI-to-GUI migration is the simplest solution. Figure 18 on page 32 shows the Systems Monitor configuration application main menu. This lists the MIB tables that can be configured on the System Monitor agents.

For migration purposes, the three tables you are interested in are:

1. Command table (SIA only). This allows arbitrary commands and shell scripts to be executed as a result of querying MIB objects.
2. File Monitor table (SIA only). This allows file status and contents to be monitored.
3. Threshold table (MLM and SLM). This polls MIB values, checks the response against a threshold and invokes an action if one is specified.

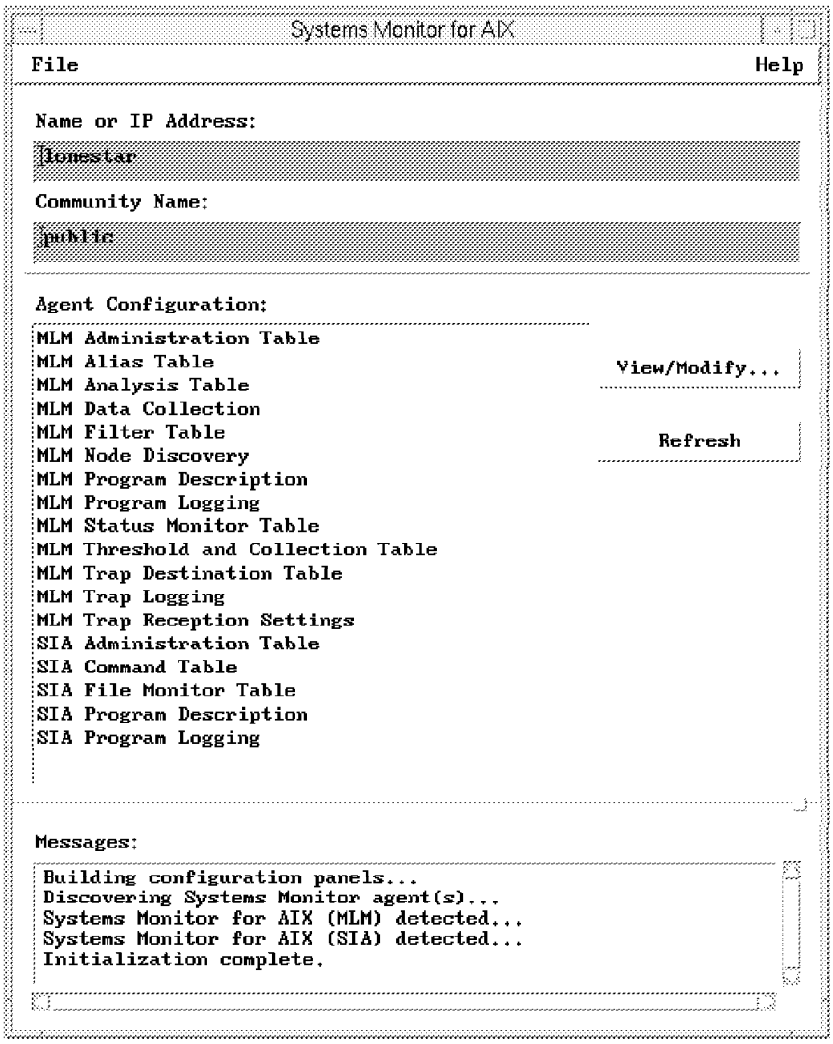



*Figure 18. An Example of the Systems Monitor Configuration Application Main Panel*

When converting Systems Monitor configurations to Sentry monitors, there are some concepts that are similar but with different names, for example:

- Command table entries define system commands to be executed. Very often they are invoked by a threshold table entry, for regular monitoring purposes. The equivalents of this combination in Sentry are the string script and numeric script monitors.
- The ″trigger when″ option in a Sentry monitor is similar to the Threshold table ″arm condition″.
- The Sentry monitoring schedule replaces the values stored in the Threshold table ″poll time″ variable.

Other concepts are not directly replicated by Sentry:

- There is no direct equivalent of the Threshold table re-arm capability, which allows an alert to be triggered only once. However, many of the Sentry monitor types have a ″changes from″ test condition, which does a similar thing in some cases.

- The Systems Monitor Command table can be used for one-time queries as well as for regular monitoring (when combined with Threshold table entries). The TME equivalent to this is provided by Task Libraries.
- There is no equivalent of the Command table shared memory monitoring function. The easiest way to replace this function may be to use an asynchronous string monitor.
- The Threshold table allows you to collect data in a log file for statistical analysis. The same thing is possible with TME 10 Distributed Monitoring, but it is not a standard feature of the product until Version 3.5 (available later in 1997).

With these variations in mind you should be able to display the configuration of your Systems Monitor tables and re-create the equivalent function using the TME GUI. In all except the smallest configurations, you will find it useful to analyze your Systems Monitor configuration using the guidelines in 4.3, “Analyzing the Migration” on page 27 before actually entering anything new.

## 4.5.2 Migration of a Larger Configuration

If you have numerous SIA, SLM and MLM agents in your current environment, then the most productive migration method may be to use the command line interface (CLI) commands. For example:

```
waddmon Unix_Sentry zombies -t ″60 minutes″ -c critical -R ″>″ 30
-p Root_ssd-region Sentry
```

This waddmon command will add a Unix_sentry monitor called zombies. The zombies monitor checks for lingering terminated processes. The -t flag is the equivalent of the Systems Monitor poll time. The -c flag defines the alert level. The defined levels are:

- Critical
- Severe
- Warning

The -R flag in this example will trigger if more than 30 terminated processes exist. The -p flag defines pop-ups to the Administrator. Sentry in this example is the profile name.

### 4.5.2.1 Migration Based on Systems Monitor Configuration Files

You could view the Systems Monitor GUI screens and then create waddmon commands that replace them. However, we would prefer to work directly from the Systems Monitor configuration files and automate the conversion where possible.

Systems Monitor stores its configurations in flat files. Each entry in the file is identified by a tag, for example, *smSiaFIleMonitorstate*. It is possible to extract certain fields from the entries and plug them directly into the waddmon command. So for example:

```
smSiaFIleMonitorstate = disabled
```

translates to:

```
waddmon +d
```

Table 3 on page 34 and Table 4 on page 34 show the equivalent waddmon parameters for SIA File Monitor table and MLM Threshold table entries.

*Table 3. SIA Command Table to waddmon Mapping*

| SIA File Monitor Table Entry | Systems Monitor Parameter | waddmon Parameter |
|---|---|---|
| smSiaFIleMonitorstate | enabled or disabled | +d -d |
| smSiaFileMonitorType | exist ,string,strDatastatus, | filechk,filesize,countstr |
| smSiaFileMonitorFullPathName | "pathname" | -a filename |
| smSiaFIleMonitorForString | "string" | -a pattern |

*Table 4. MLM Threshold Table to waddmon Migration*

| MLM Threshold Table Entry | Systems Monitor Parameter | Waddmon Parameter |
|---|---|---|
| smMlmThresholdLocalRemoteMIBVariable | "1.2.3.4.5.6.7.8.9" | Monitor Name |
| smMlmThresholdCondition | "=" ">" "<" "!=" | -R "=" "->" |
| smMlmThresholPollTIme | "5m" | -t "5 Minutes" |
| smMlmThresholdState | enabled or disabled | -d +d |
| smMlmThresholdCommandToExecute | "/usr/local/bin..." | -e "/usr/local/bin" |

The following example shows the configuration file for a threshold monitor table entry that checks whether a daemon, trapgend, is alive. The daemon will be restarted if found to be down.

```
smMlmThresholdName[Monitor_trapgend] = "Monitor_trapgend"
smMlmThresholdState = disabled
smMlmThresholdDescription = "Monitor the trapgend daemon and restart if it dies."
smMlmThresholdLocalRemoteMIBVariable = ".1.3.6.1.4.1.2.6.12.2.7.2.1.2.trapgend.*"
smMlmThresholdCondition = "doesNotExist"
smMlmThresholdPollTime = "5m"
smMlmThresholdTrapDescription = "trapgend died - auto restarting"
smMlmThresholdArmEnterprise = ".1.3.6.1.4.1.2.3.1.2.1.1.2.1"
smMlmThresholdSpecificTrap = 14
smMlmThresholdCommandToExecute = "/usr/OV/bin/trapgend"
smMlmThresholdReArmCondition = "exists"
smMlmThresholdReArmTrapDescription = "trapgend was restarted"
smMlmThresholdReArmEnterprise = ".1.3.6.1.4.1.2.3.1.2.1.1.2.1" \
smMlmThresholdReArmSpecificTrap = 14
smMlmThresholdLastResponseTime = "Thu Jan 01 00:00:00 GMT 1970"
```

A waddmon command to perform the same monitoring and restart function would be as follows:

```
waddmon Unix_sentry daemon -a trapgend -t "5 Minutes"
-c critical -R "->" "down" \
-c normal -R "->" "up" \
-p Root_ssd-region \
-e /usr/OV/bin/trapgend \
Sentry
```

There are a number of things to notice about this mapping:

1. The polling time in this case was already in minutes, which TME 10 Distributed Monitoring can use. If it had been expressed in seconds we would have had to round it to the nearest minute in the waddmon command.
2. The re-arm condition in the Systems Monitor example means that the threshold will be triggered the first time that the probe detects that trapgend is down, but not if it is still down on subsequent polls. We achieve the same thing with Sentry by using the ″Changes To″ comparator (″->″).
3. The Systems Monitor version will send a second, re-arm, event when the daemon has been restarted. In this case we do the same thing with waddmon by adding a normal response level definition that is triggered when the status changes to ″up″, but as we discuss in 5.5, “Migrating the Re-Arm Function” on page 102, this is not an ideal approach.

For the full definition of waddmon see the *Tivoli/Sentry User's Guide*.

## 4.6 Semi-Automated Profile Migration

As a logical extension of the command line migration examples above, we created a script that takes existing Systems Monitor configuration files as input and generates waddmon commands from them.

### 4.6.1 The Sample Migration Script

We provide a script that reads Systems Monitor configurations and executes waddmon commands to create equivalent Sentry profiles. The script is called migrate_sysmon_config.

To obtain the script, see Appendix B, “How to Get the Samples in This Book” on page 141.

**Disclaimer**

The script is intended only as a starting point. It will take an existing set of Systems Monitor configurations and generate a set of TME 10 Distributed Monitoring profiles from them. This is a useful shortcut, but the profiles *will* need further modification to make them do the job you want them to do.

#### 4.6.1.1 Preparing to Run the Migration Scripts

The first step is to collect all the variations of the SIA and MLM configurations from your various managed nodes and copy them into a single directory. It is not a good idea to work with the files in the normal Systems Monitor configuration directories (/var/adm/smv2/xxx/config) since this can affect the operation of a running agent.

When you have the files in one place you can create consolidated configuration files. For example, to create a configuration containing all of the SIA entries, enter:

```
grep -h smSia * > migrate.config
```

### 4.6.2 migrate_sysmon_config

The migrate_sysmon_config script performs three actions:

1. It takes the SIA command table entries and creates individual custom scripts. These are direct translations from the SIA configuration file using the smSIACommandDescription and smSIACommandGetStringAndParameters definitions.

   In some cases the scripts may invoke other scripts. Keep in mind that Command table entries run executables that are installed on the *agent* node. Sentry provides the string and numeric script monitors, which do the same thing, but specifically for shell scripts. The equivalent of the SIA Command table entry created by migrate_sysmon_config is therefore a *wrapper* script that Sentry can deploy to invoke the real executable on the target system. This is not the way that you would design a monitor from scratch.

   The scripts will all be named Migrated_{smSiaCommandName}. For example:

   `smSiaCommandName[COMMAND]`

   will be migrated to:

   `/usr/local/bin/Migrated_COMMAND`

2. The script also processes the SIA File Monitor table and attempts to create equivalent monitors, based on the mappings shown in Table 2 on page 29. In particular it maps the following File Monitor table keywords:

   - exist/doesNotExist
   - dataChange
   - statusChange (but it uses an arbitrary permissions string)
   - string

   It creates equivalent Sentry monitors for these conditions (see Table 2 on page 29 for a detailed description of the file monitor options).

3. The script processes the MLM/SLM Threshold table. The Threshold table monitors MIB objects, so the script looks for monitors of SIA MIB variables and creates an equivalent Sentry monitor where one exists. The SIA tables that are handled by the script are the following:

   - Network errors
   - File systems
   - Logged-in users
   - Paging Space
   - CPU (using the sample monitor described in 5.3, “Monitoring CPU Utilization” on page 76)
   - Command table entries (by invoking the shell scripts generated from the SIA configurations)

The script has a routine to handle MIB extensions in either dotted decimal format or combined dotted and character text, for example:

`.1.3.6.1.4.1.2.6.12.2.5.2.1.4./var`

or the equivalent dotted decimal version:

`.1.3.6.1.4.1.2.6.12.2.5.2.1.4.47.118.97.114`

are both acceptable entries for monitoring the /var filesystem.

***migrate_sysmon_config Operations:*** This script assumes that the Tivoli Management Environment is running on the machine, and that the user ID it is running under has enough authority to run waddmon in the selected policy region. The script also assumes that a Sentry profile already exists.

The script will request the following information:

- The Systems Monitor configuration file to migrate.
- The Sentry profile to which the monitors should be added.
- The directory in which custom monitoring scripts will be placed on the monitored system.

All the new monitor definitions are added in a disabled state. Poll times, which Systems Monitor allows to be specified in seconds, are rounded to the nearest minute. This script will not make any changes to the original configuration files.

Figure 19 shows a typical invocation of migrate_sysmon_config, and Figure 20 on page 38 shows the log file output from it.

```
rs600026:/tmp/migtest > migrate_sysmon_config
You are Tivoli administrator Root_ssd-region
total 176
drwxr-xr-x   2 rob      assignee    2048 Jun 05 08:34 .
drwxrwxrwt  12 root     system      3584 Jun 05 08:34 ..
-rwx------   1 root     system     25735 Jun 01 14:20 itso.config.test
-rw-r--r--   1 root     system     19384 Jun 01 07:09 kuffi
-rwxr-xr-x   1 root     system     21434 Jun 01 08:20 m_s_c
-rw-r--r--   1 rob      assignee    4449 Jun 05 08:30 mlm_plus_sia


Please input the Sysmon config name from the list above ==>mlm_plus_sia
Please input Sentry profile name in which to place migrated monitors
Note: this profile must already exist ==>migrate_test
Path to monitoring scripts on monitored system [/usr/local/bin] ==>
/tmp/migtest/mlm_plus_sia
Creating shell script for IOSTAT command table entry
Creating shell script for KERN1 command table entry
Creating countstr monitor for 'ERROR' /var/adm/smv2/log/midmand.log
Creating file checksum monitor for /etc/passwd
Monitor_Process --- Process table monitor. Creating a daemon status Sentry monitor
Monitor_var --- Creating a disk percent used monitor
lonestar_disk_space_monitor --- Creating a disk percent used monitor
baspac1_cpu_busy_monitor --- CPU utilization monitor. Using redbook monitoring collection

Migration Completed - see /tmp/migtest/migrate_sysmon.log for details
```

*Figure 19. Input for the Migration Script*

```
################## Thu Jun  5 08:34:34 EDT 1997 ########################
Creating script file: /tmp/migtest/Migrated_IOSTAT
Command:              system." iostat -t
============================================================
Creating script file: /tmp/migtest/Migrated_KERN1
Command:              smSiaCommandDescription =  "Kernel memory get of pages \
paged in" KERNEL_MEMORY: vmminfo 16 4
============================================================
SYSMON SCHEDULE====15m
SENTRY SCHEDULE====15
Creating countstr monitor for 'ERROR' /var/adm/smv2/log/midmand.log
============================================================
SYSMON SCHEDULE====30s
SENTRY SCHEDULE====1
Creating file checksum monitor for /etc/passwd
============================================================
SYSMON SCHEDULE====30s
SENTRY SCHEDULE====1
Threshold monitor for MIB: .1.3.6.1.4.1.2.6.12.2.7.2.1.2.trapgend.*
Monitor_Process --- Process table monitor. Creating a daemon status Sentry monitor
doesNotExist maps to
Daemon:       trapgend
============================================================
SYSMON SCHEDULE====5m
SENTRY SCHEDULE====5
Threshold monitor for MIB: .1.3.6.1.4.1.2.6.12.2.5.2.1.4.47.118.097.114
Monitor_var --- Creating a disk percent used monitor
Trigger value: 95
Condition:     >
Filesystem:    /var
============================================================
SYSMON SCHEDULE====5m
SENTRY SCHEDULE====5
Threshold monitor for MIB: .1.3.6.1.4.1.2.6.12.2.5.2.1.4.*
lonestar_disk_space_monitor --- Creating a disk percent used monitor
Trigger value: 95
Condition:     >
Redbook sample monitor for all filesystem util% added
============================================================
```

*Figure 20. Log File from migrate_sysmon_config*

Figure 21 on page 39 shows an example of a TME 10 Distributed Monitoring profile created by migrate_sysmon_config.

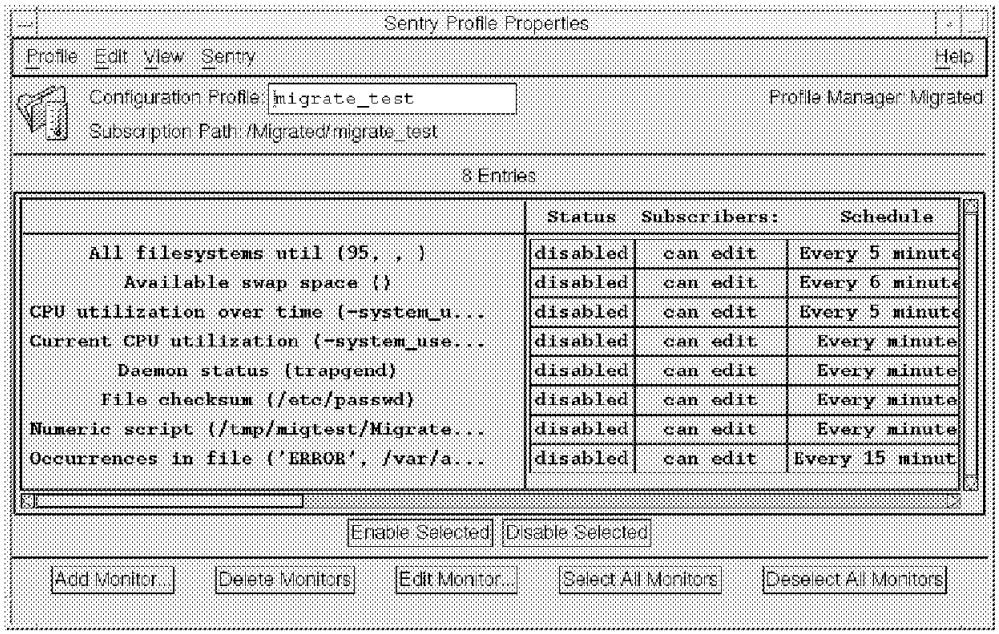



*Figure 21. Sentry Profile Generated by migrate_sysmon_config Script*

Figure 22 shows a migrated file monitor configuration and Figure 23 on page 40 shows a monitor that invokes a migrated Command table entry.

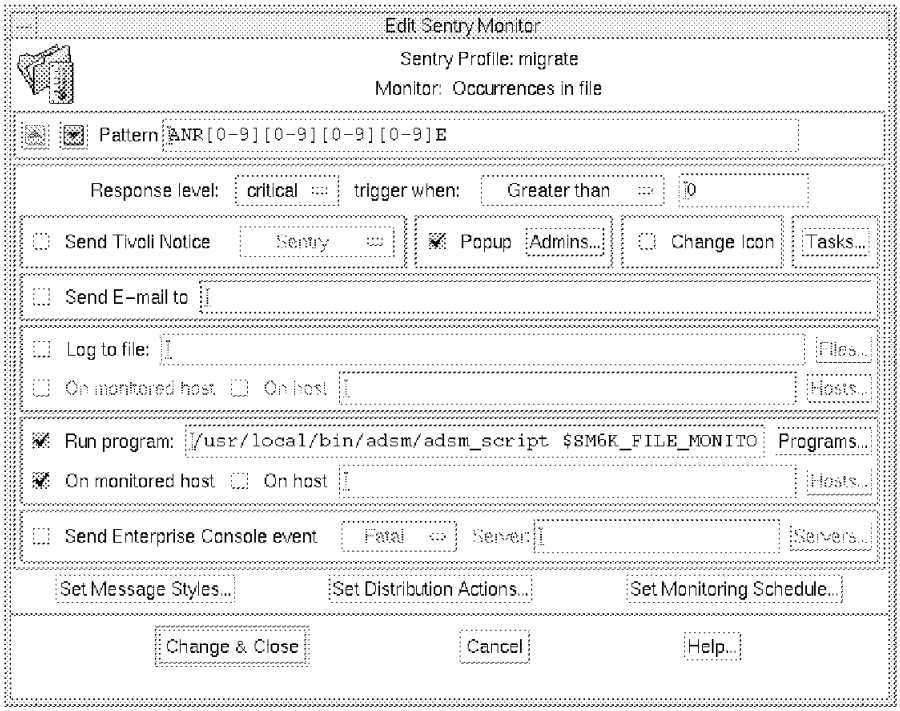



*Figure 22. An Example of a Migrated File Monitor Table*

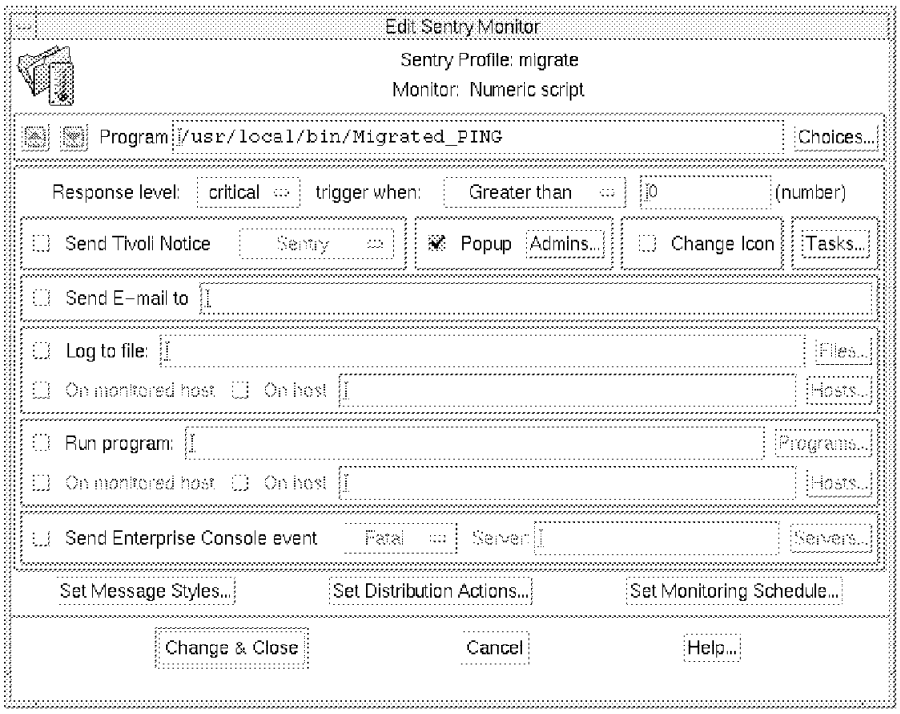



*Figure 23. An Example of a Migrated Command Script*

In the logfile you will see information about which configuration entries are migrated and which are not. If you want to know more about the MIB extensions mentioned in the logfile, use the TME 10 NetView MIB Browser application. For example, the object .1.3.6.1.4.1.2.6.12.2.4.3.* is the percent of kill threshold to free space variable from the SIA MIB. This monitor is not directly available in Sentry, so it cannot be migrated. Figure 24 on page 41 shows the MIB Browser display for this.

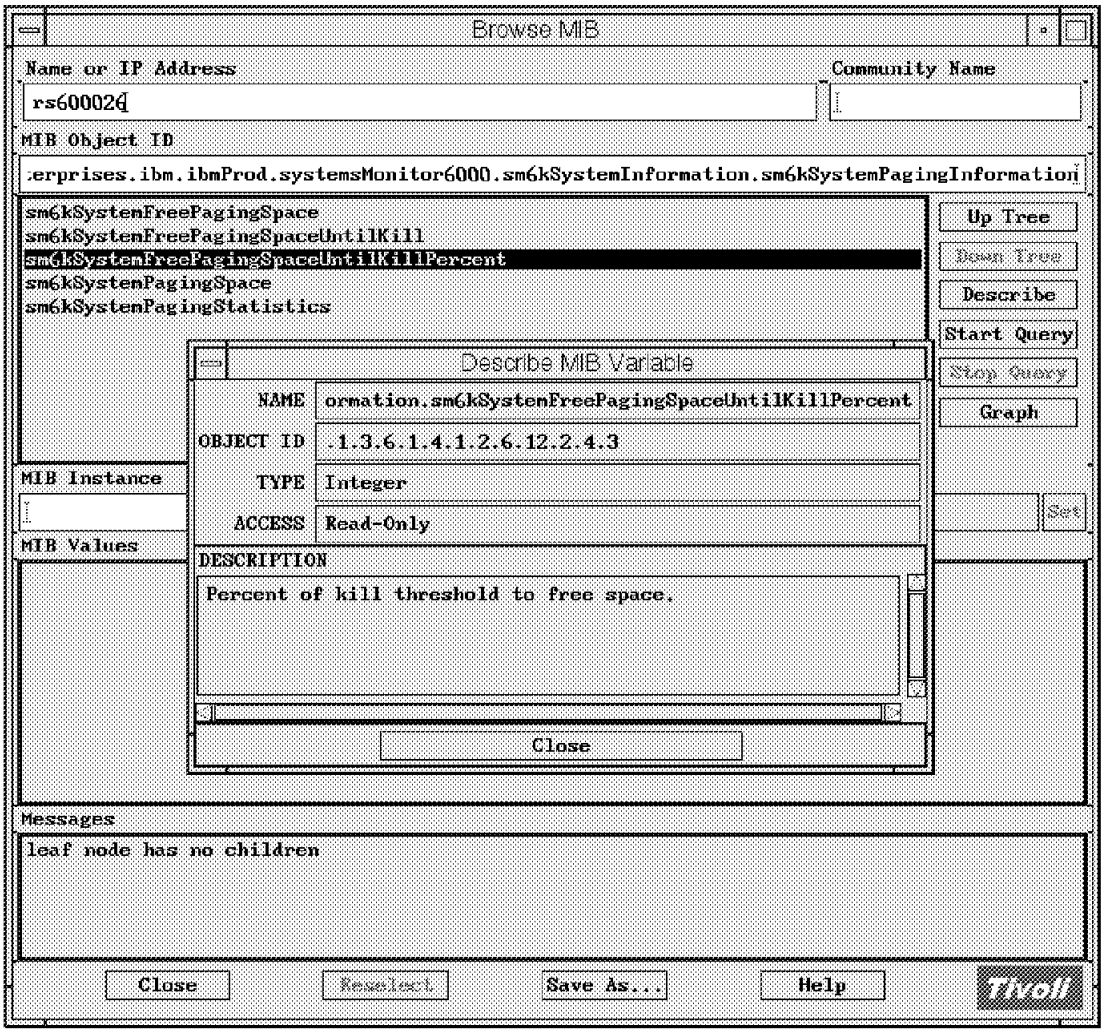



*Figure 24. Using the MIB Browser to Check MIB Extension Descriptions*

### 4.6.3 After Running the Migration Script

Once you have run the script against your SIA, MLM and SLM configuration files you need to check all of the monitors in the profile. Make sure that everything you expected has been migrated. Some monitors may need to be added manually.

There may also be monitors that you did not want to generate. Delete anything that should not be there. A certain amount of tidying is required. For instance, you may wish to rename the migrated scripts to more meaningful names. Don't forget to change the monitor entry to correspond with each script name.

# Chapter 5. TME 10 Distributed Monitoring Examples

In this chapter we show a number of examples of different techniques using TME 10 Distributed Monitoring. Most of these examples are directly related to the question of migration from Systems Monitor, but some of them simply show solutions to monitoring problems that the authors have encountered in their practical experience.

The examples in this chapter are as follows:

- 5.1, “File System Monitoring Example” on page 44 is a simple example to show a migrateable monitor (category 1). We then go on to show the different ways that the events generated by the monitor can be displayed, using the TME desktop, T/EC and TME 10 NetView.
- 5.2, “Print Subsystem Monitoring” on page 66 shows some examples of techniques to monitor the print subsystem (lpd daemon) including automatic restart actions. In this part we also use TME 10 tasks and the TME 10 Task Library Language.
- 5.3, “Monitoring CPU Utilization” on page 76 shows an example of a locally written monitoring collection for checking CPU utilization. This is an example of a category 2 monitor, or one that does not migrate directly from Systems Monitor to TME 10 Distributed Monitoring. In this part we also use the MCSL language to create a monitoring collection and a monitor.
- 5.4, “File Monitoring Examples” on page 93 shows examples of some more complex file monitoring requirements, including handling files with variable filenames.
- 5.5, “Migrating the Re-Arm Function” on page 102 shows a possible approach to re-creating the re-arm capability of Systems Monitor MLM, which is not normally supported by TME 10 Distributed Monitoring.
- 5.6, “Advanced Process Monitoring” on page 107 shows a locally created monitoring collection that allows you to keep track of multiple, related processes on a system. In this part we use the MCSL language again to create a monitor.
- 5.8, “Generic File System Monitoring” on page 115 shows how to monitor multiple file systems on an AIX system.
- 5.9, “SNMP Proxy Forwarding” on page 118 describes the proxy monitoring capability of TME 10 Distributed Monitoring and shows how to apply it to an SNMP-managed system.
- 5.7, “Hardware Alerting from AIX Error Report” on page 112 shows how to use a TME 10 Distributed Monitoring asynchronous monitor to handle events written to the AIX error logging facility.
- 5.10, “Migrating Data Collection” on page 124 discusses how to use TME 10 Distributed Monitoring 3.5 to collect historical data and shows ways to extract the data for processing.

## 5.1 File System Monitoring Example

Both, Systems Monitor and TME 10 Distributed Monitoring provide monitors for checking the free space within a UNIX file system. If you refer to the table in

Appendix A, “Mapping SIA MIB Objects to Sentry Monitoring Collections” on page 137 you will see that these monitors fall into category 1. That is, there is a direct mapping between the Systems Monitor and Sentry functions.

We use this monitor as a simple example of how to create a TME 10 Distributed Monitoring definition with equivalent function. Then we will explore the different facilities for reporting events that the TME 10 environment offers, namely:

- TME desktop displays
- Routing events to the TME 10 Enterprise Console
- Routing events to TME 10 NetView

### 5.1.1 Migration from Systems Monitor

As we described in 4.5, “Practical Migration Techniques” on page 31, we could use a semi-automated approach to migrate this monitor. The migrate_mlm_config shell script will create an equivalent Sentry definition given an MLM configuration file. Alternatively, you can perform a manual migration by looking at the Systems Monitor GUI and creating an equivalent using the Sentry GUI. We will show the former approach.

#### 5.1.1.1 Defining the Monitor in Systems Monitor MLM

The monitor we define will check the size of the /var file system. If it exceeds 90% utilized we will generate a threshold event. We define the MLM monitor using the Systems Monitor graphical configuration interface (smconfig) to define the entry. Note that in this case the MLM threshold table is monitoring a MIB value provided by the SIA.

To start the Systems Monitor configuration interface enter `smconfig` on an AIX command line or select it from the TME 10 NetView tools menu. When it has initialized, select **MLM Threshold and Collection Table**, then click on **Add/Copy** in the resulting panel. Figure 35 on page 75 shows how to fill in the monitor details for our file system monitoring example. You can see that we have defined that a threshold should be triggered (or, in Systems Monitor terms, *armed*) whenever the /var filesystem is more than 90% utilized. We have asked for the utilization to be checked every five minutes (defined in the Poll Time field).

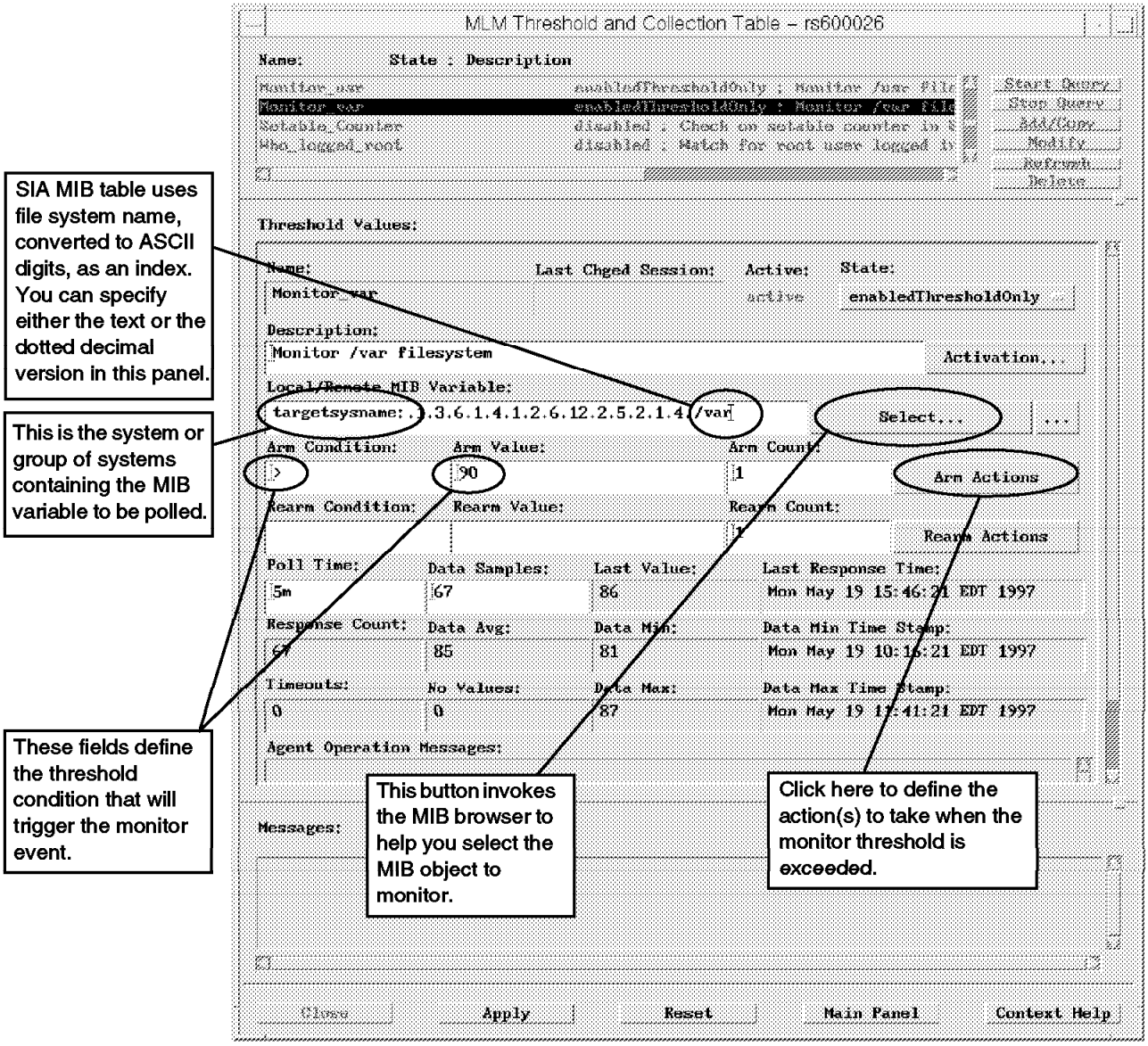



*Figure 25. Defining a File System Monitor Using MLM*

Having set up the threshold definition itself, the next step is to define what should happen when it is exceeded. We do this by clicking on **Arm Actions**. Systems Monitor provides two possibilities: you can cause an SNMP trap to be sent to the managing system and/or you can cause a command to be executed. Note that any automated command will be executed on the MLM system, *not* on the system that is being monitored. Figure 26 on page 46 shows the action definition screen.

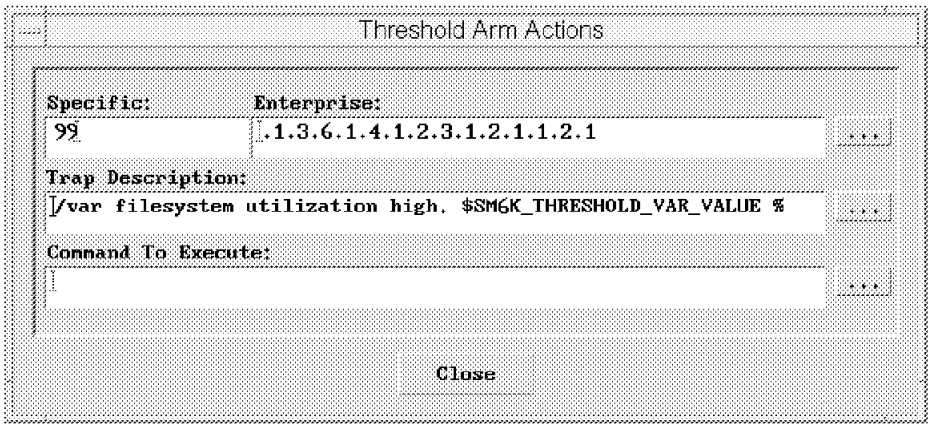



*Figure 26. Defining Threshold Actions for MLM*

In this case we have specified that a trap should be sent, under the Systems Monitor enterprise ID (the default) and with specific trap ID 99. Notice that we have used an MLM-provided environment variable, $SM6K_THRESHOLD_VAR_VALUE, in the trap description. This will be replaced by the value that caused the threshold to be triggered, that is to say the actual utilization of the file system. We have not asked for a command to be executed in this case.

### 5.1.1.2 Defining the Monitor in TME 10 Distributed Monitoring

Sentry monitors are defined as profiles within a profile manager, which is created in the context of a policy region. We assume that you are familiar with creating policy regions, profile managers and profiles. Refer to Chapter 3, “TME 10 Distributed Monitoring in Detail” on page 15 for an explanation of these terms. To create the monitor itself, double click the profile manager icon and select **Create**, followed by **Profile** from the menu bar. Select **SentryProfile** from the list and give the profile a suitable name. Figure 27 on page 47 shows the steps involved in creating a new profile, starting at the TME desktop. Once you have created a profile it can contain any number of monitors. You should group monitors together in a logical way, for example, by collecting all monitors relating to a specific type of resource or application.

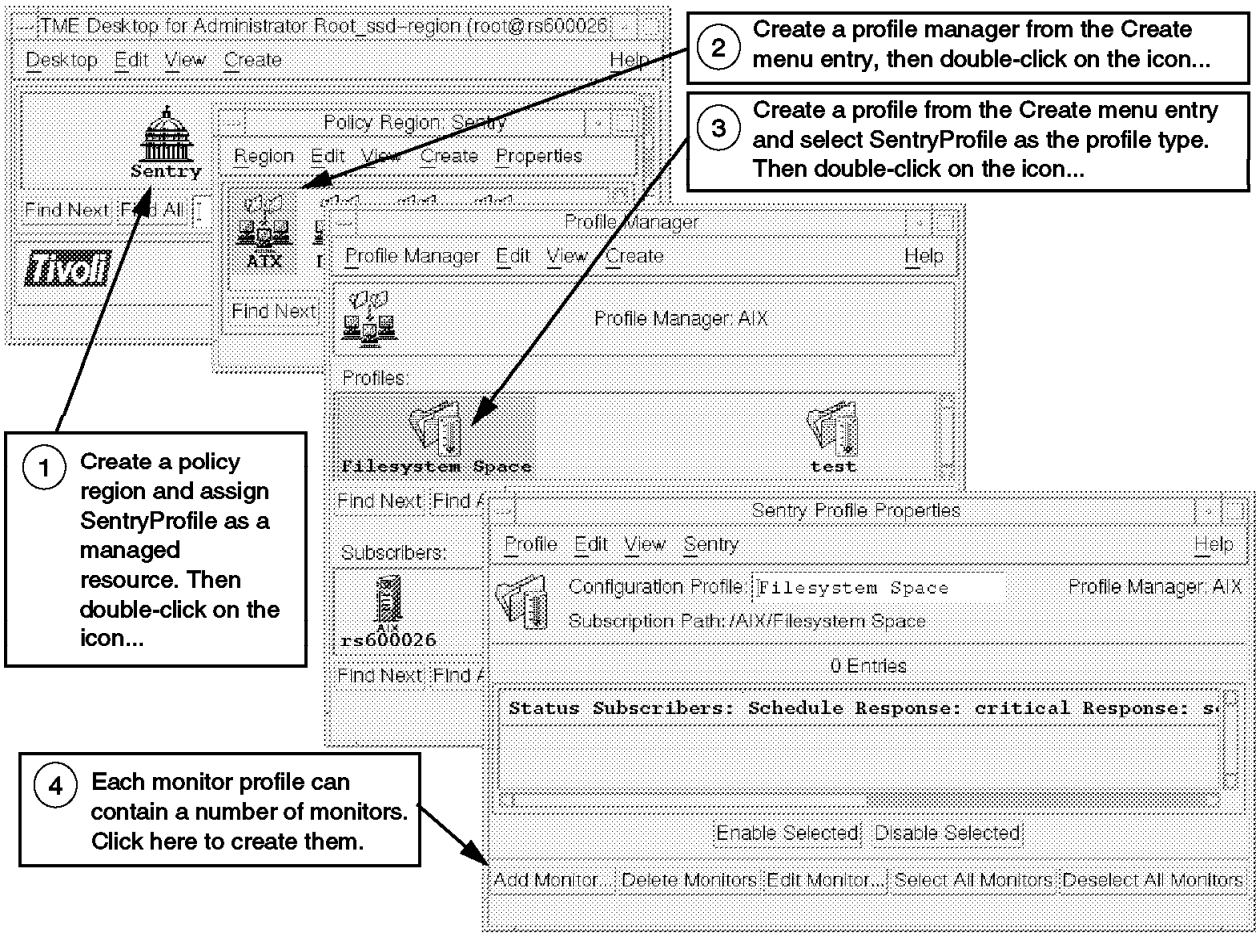



*Figure 27. Steps to Create a Sentry Profile*

To create our disk space monitor we would normally click on **Add Monitor** and then specify a monitoring collection of Universal and a monitor type of Disk space used %. However, we chose to convert the Systems Monitor example directly to a Sentry monitor, using the migrate_sysmon_config script described in 4.6, “Semi-Automated Profile Migration” on page 35.

### 5.1.1.3 Using the migrate_sysmon_config Script

To perform the migration, we first copied the MLM configuration file from its default location in directory /var/adm/smv2/mlm/config into a working directory, trimmed it so that it only included the monitor we just created, and then invoked the command. The command dialog is shown in Figure 28 on page 48 and the resulting monitor definition is shown in Figure 29 on page 48.

```
rs600026:/tmp/migtest > /:/project/sentry/samples/migration_scripts/migrate_sysmon_config
You are Tivoli administrator Root_ssd-region
total 128
drwxr-xr-x   2 rob      assignee    2048 Jun 05 09:33 .
drwxrwxrwt  12 root     system      3584 Jun 05 09:33 ..
-rwx------   1 root     system       862 Jun 05 09:31 itso.config.test
-rw-r--r--   1 root     system     19384 Jun 01 07:09 kuffi
-rwxr-xr-x   1 root     system     21434 Jun 01 08:20 m_s_c
-rw-r--r--   1 rob      assignee    4449 Jun 05 08:30 mlm_plus_sia


Please input the Sysmon config name from the list above ==>itso.config.test
Please input Sentry profile name in which to place migrated monitors
Note: this profile must already exist ==>migrate_test
Path to monitoring scripts on monitored system [/usr/local/bin] ==>
/tmp/migtest/itso.config.test
Monitor_var --- Creating a disk percent used monitor

Migration Completed - see /tmp/migtest/migrate_sysmon.log for details
rs600026:/tmp/migtest > cat migrate_sysmon.log
################## Thu Jun  5 09:33:25 EDT 1997 ##########################
SYSMON SCHEDULE====5m
SENTRY SCHEDULE====5
Threshold monitor for MIB: .1.3.6.1.4.1.2.6.12.2.5.2.1.4./var
Monitor_var --- Creating a disk percent used monitor
Trigger value: 90
Condition:     >
Filesystem:    /var
============================================================
```

*Figure 28. Migration of an MLM Configuration Entry with migrate_sysmon_config*

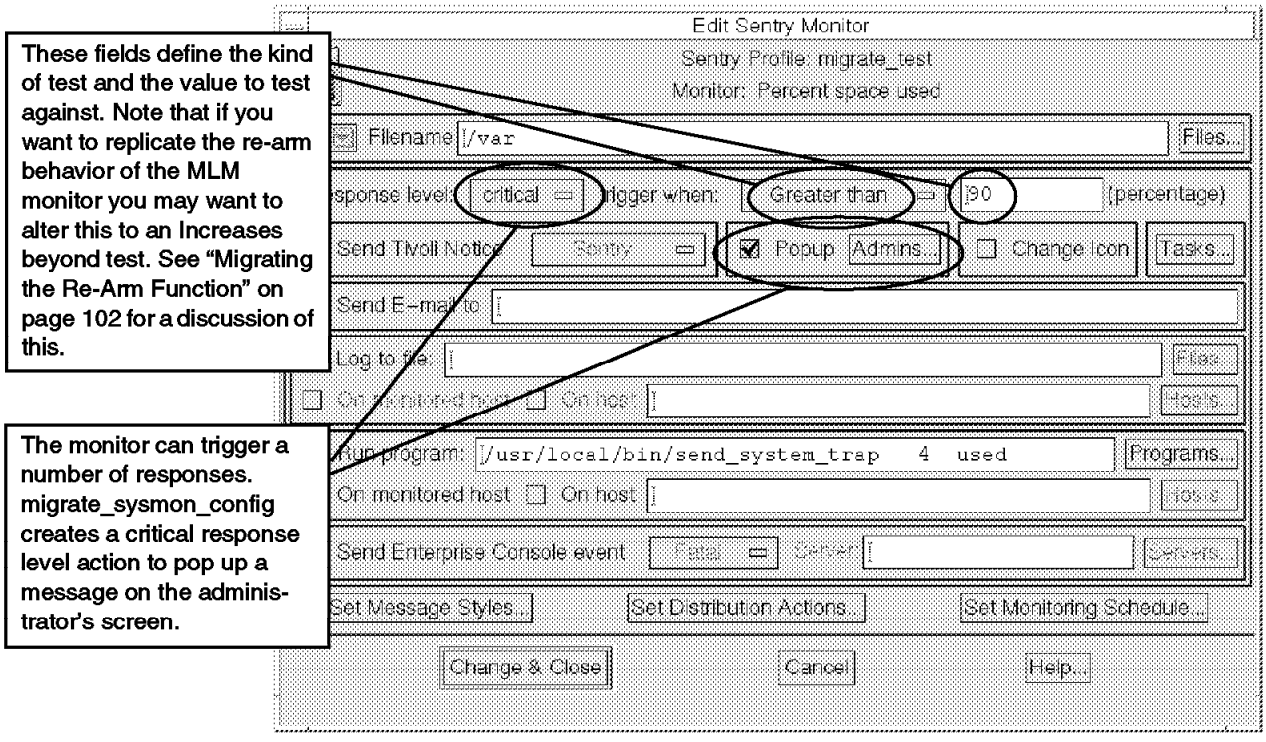



*Figure 29. Monitor Created by migrate_sysmon_config*

Note that there are a larger number of actions available for the Sentry monitor compared to the MLM example. The migration script simply defines a pop-up action, so you may need to modify this monitor before enabling it and distributing it to the monitored nodes.

#### 5.1.1.4 Distributing the Two Monitors

We have now created two monitors, one using MLM, the other using Sentry. They perform an identical function, that is to say, they both monitor file systems utilization at five-minute intervals and raise an alarm if it exceeds 90%. Although they do the same thing, there are a number of important differences between them:

1. The MLM monitor executes on a separate system from the machine being monitored, using SNMP to poll across the network. Sentry monitoring takes place locally on the target machine.
2. The MLM monitor can act on either a single system or on a group of systems (by replacing the system name with an alias). When the monitor definition is saved, it will take effect immediately. With Sentry, by contrast, the target systems must first be subscribed to the monitoring profile and the profile must be distributed before it becomes active.
3. The alarms from MLM appear as events on the SNMP manager display (for example, the TME 10 NetView control desk). Sentry has a number of different ways to generate event messages, in addition to the pop-up option that we have shown.

We will now explore some of the other options for displaying events generated by a Sentry monitor.

### 5.1.2 Different Ways to Display Monitor Events from TME 10 Distributed Monitoring

In addition to creating a pop-up message, you can highlight a Sentry threshold event on the TME desktop by generating a notice or updating an indicator icon. We do not go into the details of these methods here; refer to the TME 10 Distributed Monitoring documentation for details about them. In this section we show two more complex examples of event message generation:

- Sending Sentry events to TME 10 Enterprise Console
- Using automation to send Sentry events to TME 10 NetView

### 5.1.3 Sending Events to TME 10 Enterprise Console

In many cases where you have established TME 10 as the solution for enterprise systems management, you will have a T/EC server installed. The T/EC server acts as the central point where events are correlated and automated actions are initiated, so you will want to send at least some of the TME 10 Distributed Monitoring threshold events to T/EC.

However, before you can use T/EC in combination with TME 10 Distributed Monitoring, there are certain tasks you have to perform to configure T/EC for that purpose. These tasks may vary, depending on your specific environment, but the following list is a summary of the complete procedure:

1. Define a TME administrator that uses a T/EC event console
2. Create or modify a T/EC rule base
3. Import the TME 10 Distributed Monitoring event definitions into the rule base
4. Add a new event source for TME 10 Distributed Monitoring
5. Add a new event group
6. Create an event console

7. Assign event groups to an event console

We will go through each of these steps in turn.

### 5.1.3.1 Adding a TME Administrator

You can either use an existing TME administrator that uses the new T/EC event console or create a new one. In our example we create a new TME administrator called Sentry_Administrator.

Open the TME 10 desktop and double-click on the **Administrators** icon. The screen shown in Figure 30 will appear.

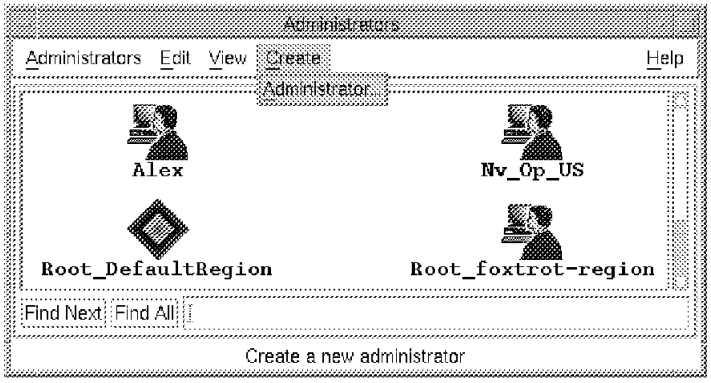



*Figure 30. Create Sentry_Administrator*

Select **Create** from the menu bar and then **Administrator...** from the pull-down menu. In the window that will appear, enter Sentry_Administrator as the Icon label.

Select appropriate Logins and Notice Group Subscriptions for the new administrator. The TMR Roles and Resource Roles can be used to control the new administrator's access to T/EC. When finished, select the **Create & Close** button.

### 5.1.3.2 Creating a T/EC Rule Base

A T/EC rule base holds two types of information:

- Event classes describe the format of events that are sent to T/EC. Each event adapter provides class definitions describing the events it wants to send to T/EC. These definitions are usually supplied in a .baroc file with the event adapter.
- Event rules describe how to handle certain events or combinations of events. Usually, these rules are associated with automated actions to be performed when events arrive. While each event adapter has to supply an event class definition for the events it wants to send it does not necessarily have to also supply standard event rules.

After the installation of T/EC there is a standard rule base called Default. You can extend this rule base with new events and rules. However, it is always wise to create a new rule base.

Since there can be only one active rule base per event server, you usually create a new rule base, copy an existing one over it and then add the event

classes and rules to the new rule base. Using that procedure, you can combine event classes and rules from different sources.

To create a new rule base double-click on the **EventServer** icon on the TME desktop or use the right mouse button and select **Rule Bases...** from the pop-up menu as shown in Figure 31.

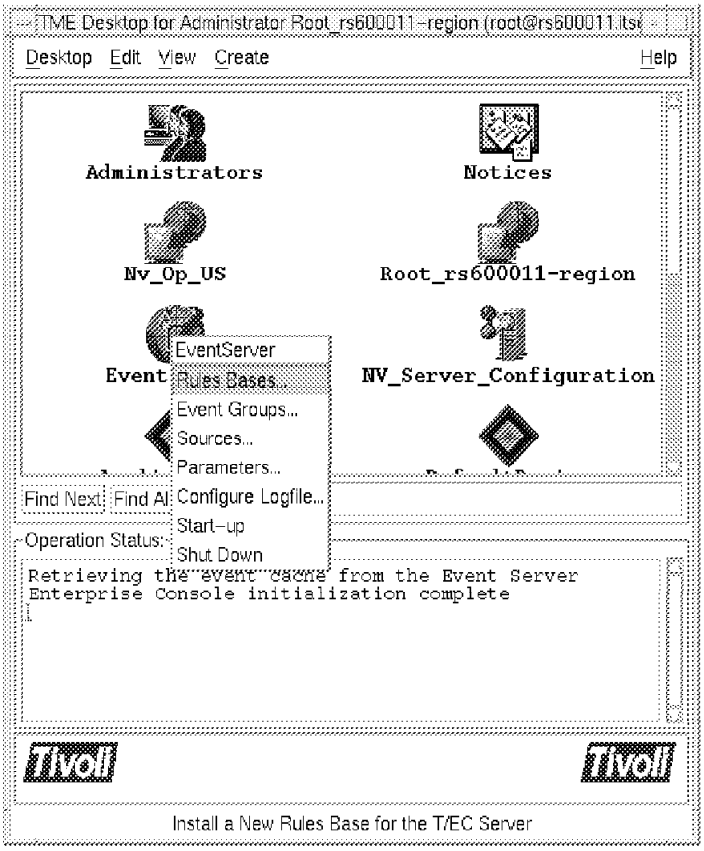



*Figure 31. TME Desktop Window*

The Event Server Rule Bases window will appear. Select **Create** from the menu bar and then **Rule Base...** from the pull-down menu. The following window will appear:

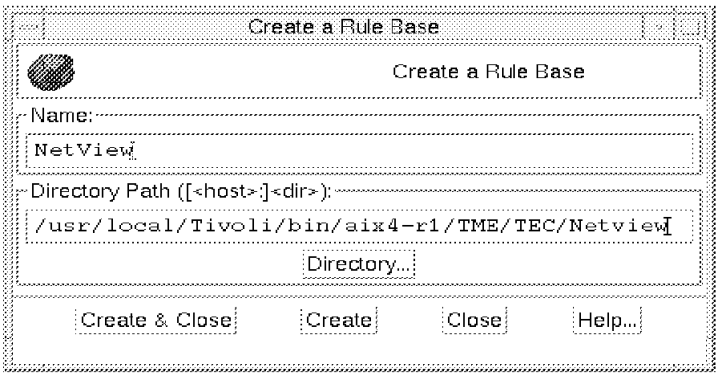



*Figure 32. Create Rule Base*

Enter a Name for the new rule base and a Directory Path where to store it. The directory that you specify here will be updated with a number of subdirectories containing the event classes and rules for the new rule base.

Select the **Create & Close** button to create the new rule base. This will return you to the Event Server Rule Bases window where you will find an icon that has been added to represent the new rule base as shown in Figure 33.

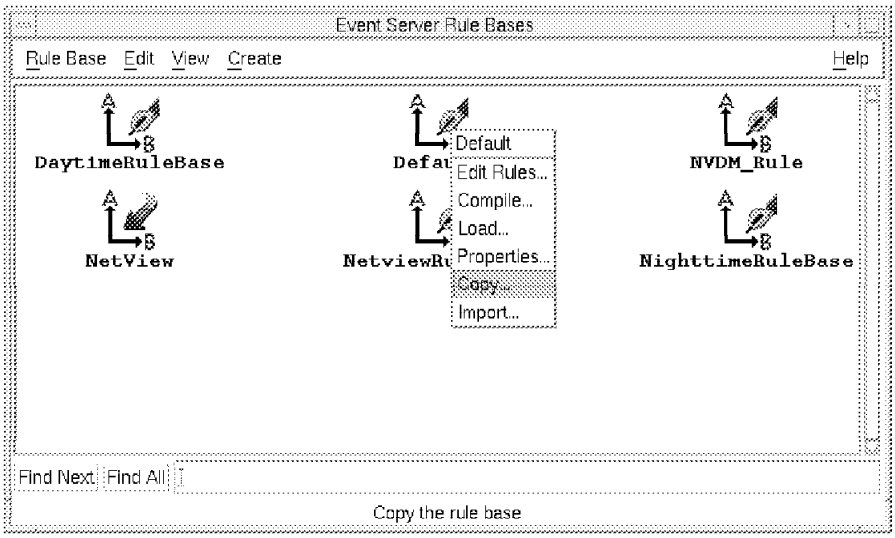



*Figure 33. Copy Default Rule Base to NetView Rule Base*

The rule base that we have just created is empty. The first thing to do is to copy the contents of the Default rule base to the new rule base. To do this, select the rule base you want to copy (in our case the Default rule base), hold down the right mouse button and select **Copy...** from the pop-up menu. The window shown in Figure 34 will appear.

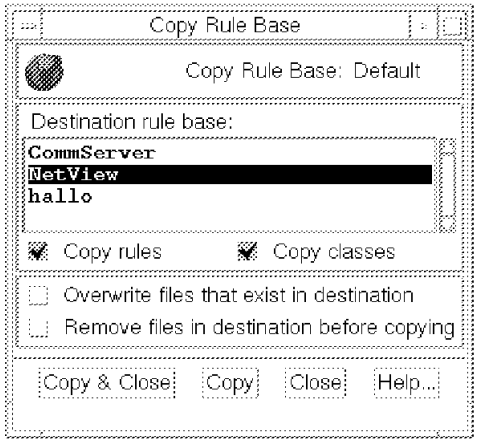



*Figure 34. Copy Rule Base*

Select **NetView** (the name of the new rule base) in the Destination rule base list and check the **Copy rules** and **Copy classes** check boxes. Then click on **Copy & Close** to copy the Default rule base onto the new rule base.

#### 5.1.3.3 Importing the TME 10 Distributed Monitoring Event Definitions

TME 10 Distributed Monitoring can generate events and send them to the T/EC, so it contains an event adapter. Unlike most other adapters, which are separate programs, the TME 10 Distributed Monitoring event adapter is integrated in the Sentry application.

When TME 10 Distributed Monitoring is installed, .baroc files that define its event classes and rules are placed in the directory /usr/local/Tivoli/bin/generic/SentryMonitors.

To have TME 10 Distributed Monitoring send events to T/EC, it is necessary to import the following class definition files into the rule base:

- Sentry.baroc
- tivoli.baroc
- universal.baroc

Select the newly created rule base by holding down the right mouse button and select **Import...** from the pop-up menu as shown in Figure 35.

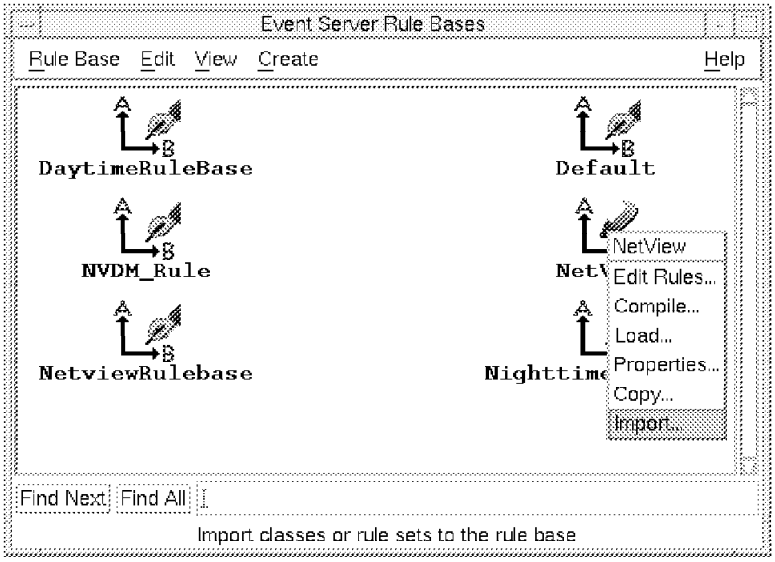



*Figure 35. Importing the Sentry Event Classes*

The Import Into Rule Base window will appear as shown in Figure 36 on page 54. Notice that this dialog has two parts to it, one for importing classes into the rule base and the other for importing rules. Be careful that you do not select the wrong section. If you import a BAROC class as a rule the rule base will not subsequently compile successfully.

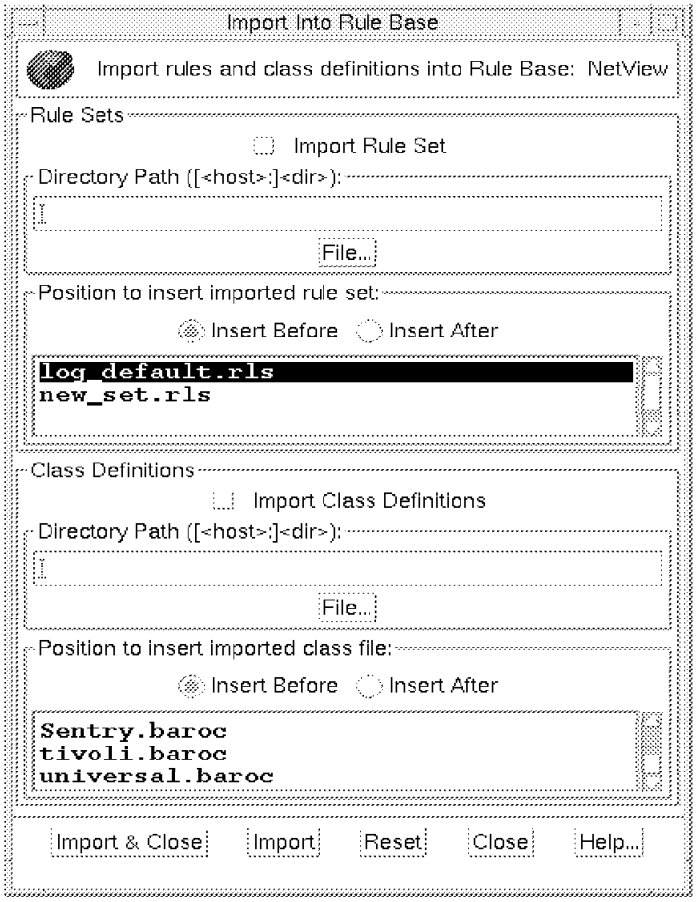



*Figure 36. Import the Sentry Class Definition Files*

Select the **Import Class Definitions** button and specify the directory where the TME 10 Distributed Monitoring .baroc files are located in the Directory Path field. By default this is the directory /usr/local/Tivoli/bin/generic/SentryMonitors.

Select the files mentioned before and then click on **Import & Close**. This will return you to the Event Server Rule Bases window.

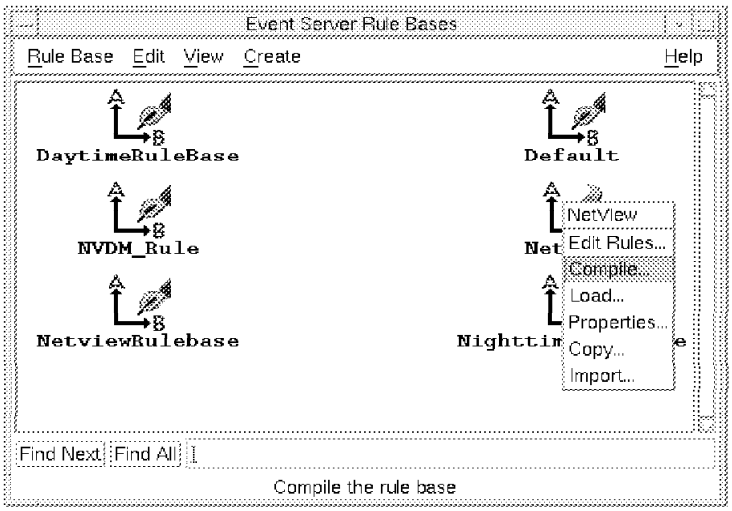



*Figure 37. Compile the Rule Base*

When you add definitions to a rule base, you should always compile the rule base. You do this by selecting the rule base icon with the right mouse button pressed and then selecting **Compile...** from the pop-up menu.

We now have created a new rule base, extended it with the definitions for TME 10 Distributed Monitoring and compiled it. However, normally the rule base is not active yet. There can only be one rule base active at one time. To activate the new rule base, select it with the right mouse button and then select **Load...** from the pop-up menu as shown in Figure 38.

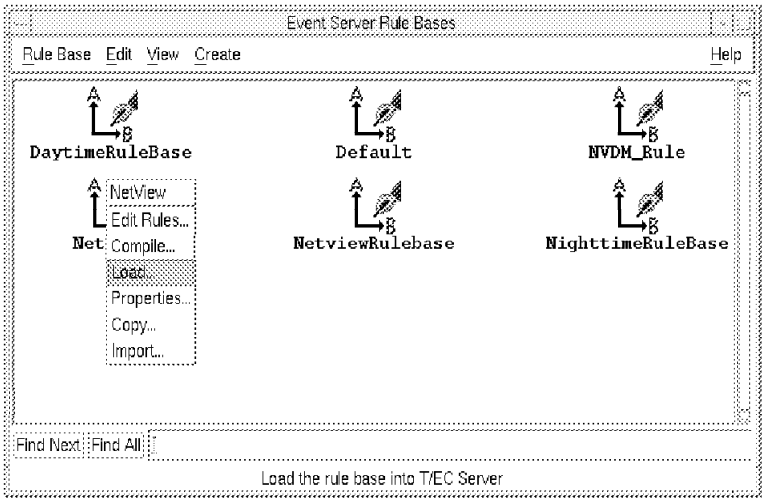



*Figure 38. Load the Rule Base*

When you load the rule base you will be prompted whether to load it immediately or to wait until the event server is restarted:

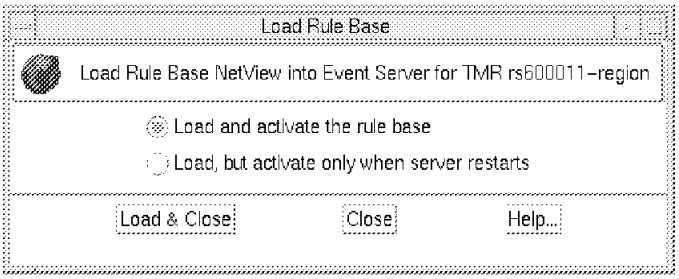



*Figure 39. Loading the Rule Base*

In this case we click on **Load and activate the rule base** and then select **Load & Close**. However, when you add new classes, the new rule base will not be active until the T/EC event server is restarted (unlike new rules, which take effect immediately). Do this using the wstopesvr and wstartesvr commands or the context menu on the Event Server icon.

### 5.1.3.4 Adding a New Event Source for TME 10 Distributed Monitoring

We have now customized the input to the T/EC server so that it will recognize Sentry events correctly. Next we have to configure the output from the server, so that the events will appear on event consoles. We do this by adding a new event source for TME 10 Distributed Monitoring that we can then assign to event consoles.

To add a new event source go to the TME desktop, click on the **Event Server** icon with the right mouse button and select **Sources** from the pop-up menu. The T/EC Source List window will appear, as shown in Figure 40 on page 56.

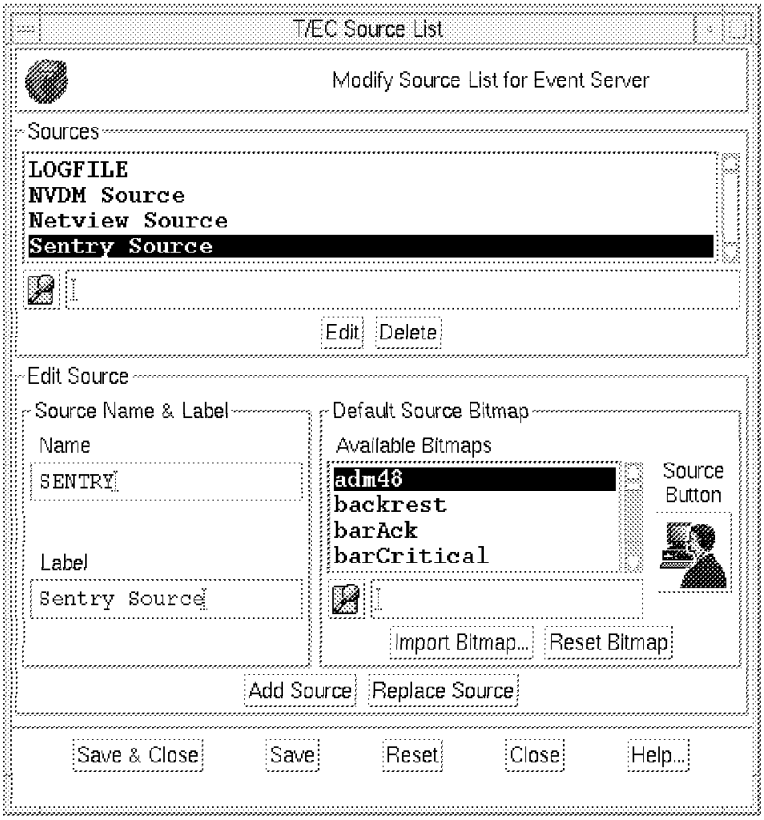



*Figure 40. Adding Source*

Enter Sentry Source into the Source Label field and enter SENTRY as the Source Name. This source field value has to match the source slot in the BAROC class definition of the event type. Then click on **Add Source** followed by **Save & Close**.

### 5.1.3.5 Adding a New Event Group

You may not want every administrator to see all of the Sentry events. For example you may want to subdivide them by different monitored system type or severity. T/EC provides you with event groups as a means to group events from certain sources according to predefined filter criteria.

We add a new event group for events coming from the TME 10 Distributed Monitoring application. Select the **EventServer** icon with the right mouse button and then select **Event Groups...** from the pop-up menu as shown in Figure 41 on page 57.

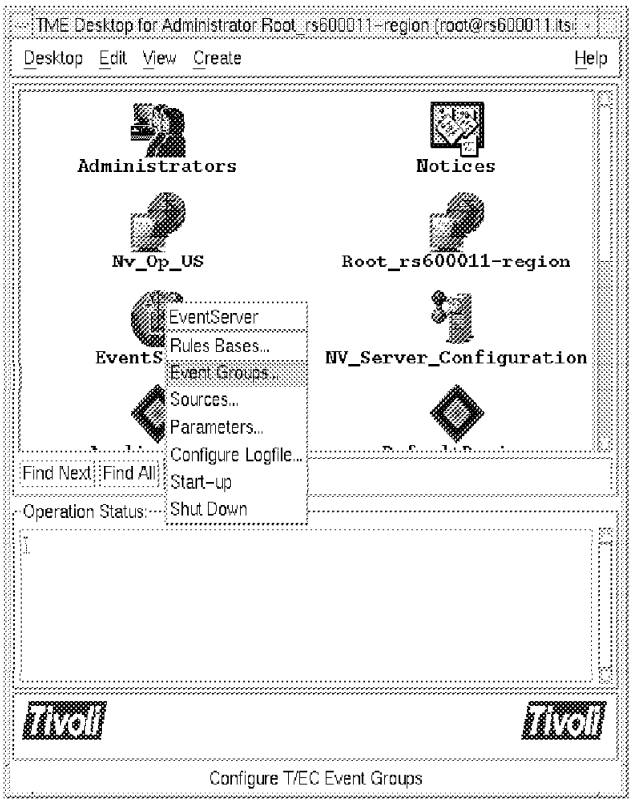



*Figure 41. Adding an Event Group*

The Event Group Management window will appear. This allows you to define a number of different groups. Select **Event Group** from the menu bar and then **New...** from the pull-down menu.

The New Event Group window will appear, as shown in Figure 42.

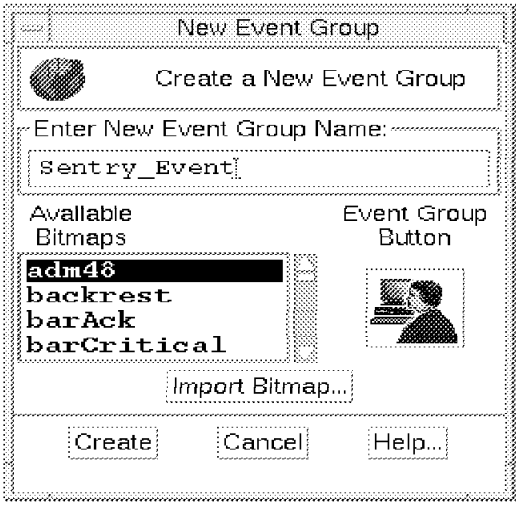



*Figure 42. Add Sentry Event Group*

Enter a name for the new event group in the Enter New Event Group Name field and then click on **Create**. The screen shown in Figure 43 on page 58 will appear.

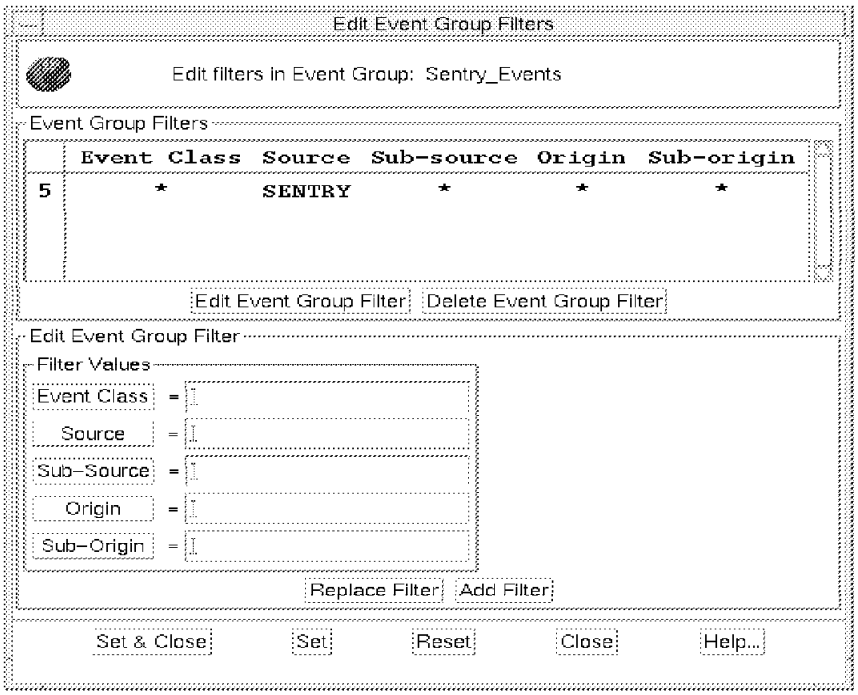



*Figure 43. Setting Event Group FIlters*

Enter SENTRY in the entry field next to the **Source** button and leave all other fields blank. Then click on **Add Filter**. This simple filter definition will accept all events that come from the SENTRY source. Click on **Set & Close**.

### 5.1.3.6 Creating an Event Console

If you already have an event console for the administrator that is to monitor TME 10 Distributed Monitoring events, then you can skip this step.

We create an event console for monitoring TME 10 Distributed Monitoring events. To do so, open the TME desktop of the administrator for whom the new event console shall be created. In our case this is Sentry_Administrator. If you have sufficient authority, you can get to the desktop for a specific administrator by double-clicking the **Administrators** icon in the TME desktop and then double-clicking the icon for the administrator.

In the desktop window for the administrator, select **Create** from the menu bar and then **Event Console...** from the pull-down menu. Select a host where the event console shall be executed, and then click on **Create** (the event console will normally run on the T/EC server machine, but if you install the T/EC code on another managed node it can run there instead).

This will create a new event console, represented by an event console icon on the administrator's desktop.

### 5.1.3.7 Assigning Event Groups to the Event Console

In order to have events from Sentry arrive at our event console, we need to assign the event group that we have created in 5.1.3.5, "Adding a New Event Group" on page 56. Select the event console icon with the right mouse button and then select **Assign Event Groups** from the pop-up menu (see Figure 44 on page 59).

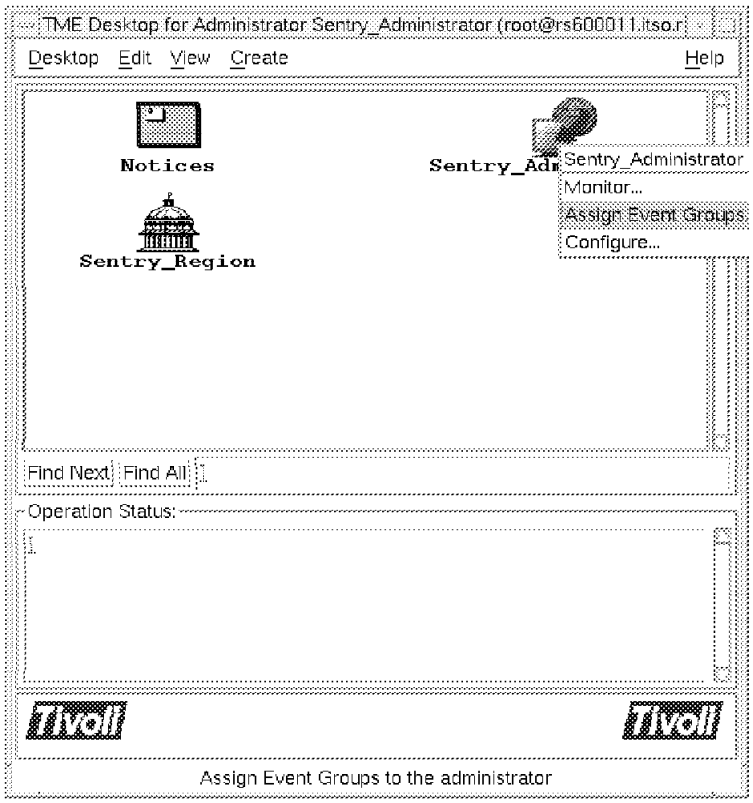



*Figure 44. Assigning Event Group to Console*

The Assign Event Groups window will appear, as shown in Figure 45.

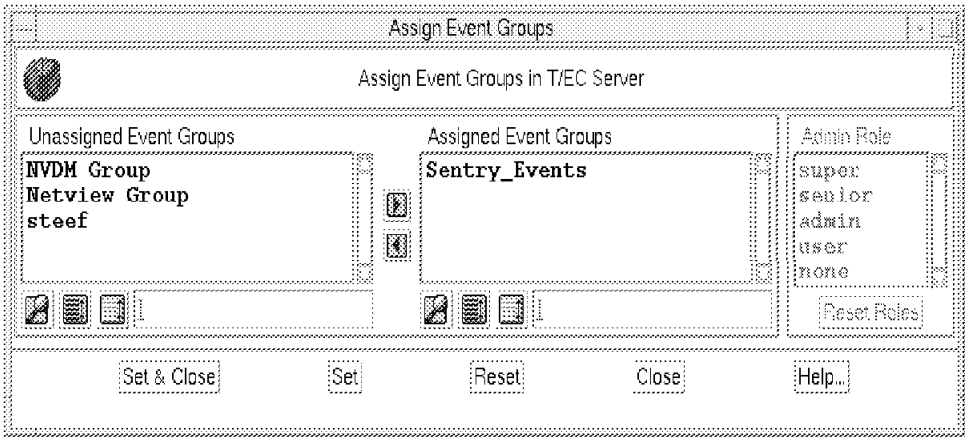



*Figure 45. Assigning Event Group to Sentry_Admin Console*

Select the Sentry_Events group that we previously created and then click on the right arrow button to move it to the Assigned Event Groups window. Then select it again and select the roles that the administrator has for event display and manipulation. If you want the administrator to be able to close events you must assign the senior role. When finished, click on **Set & Close**.

### 5.1.3.8 Receiving T/EC Events from Sentry

Finally, your event console is ready to receive events from Sentry. Start the TME desktop using the TME administrator created in 5.1.3.1, “Adding a TME Administrator” on page 50 and then double-click on the event console icon.

Make sure that the file system monitors are active. When the file system monitors are triggered, you will see events from Sentry arriving at your event console as shown in Figure 46.

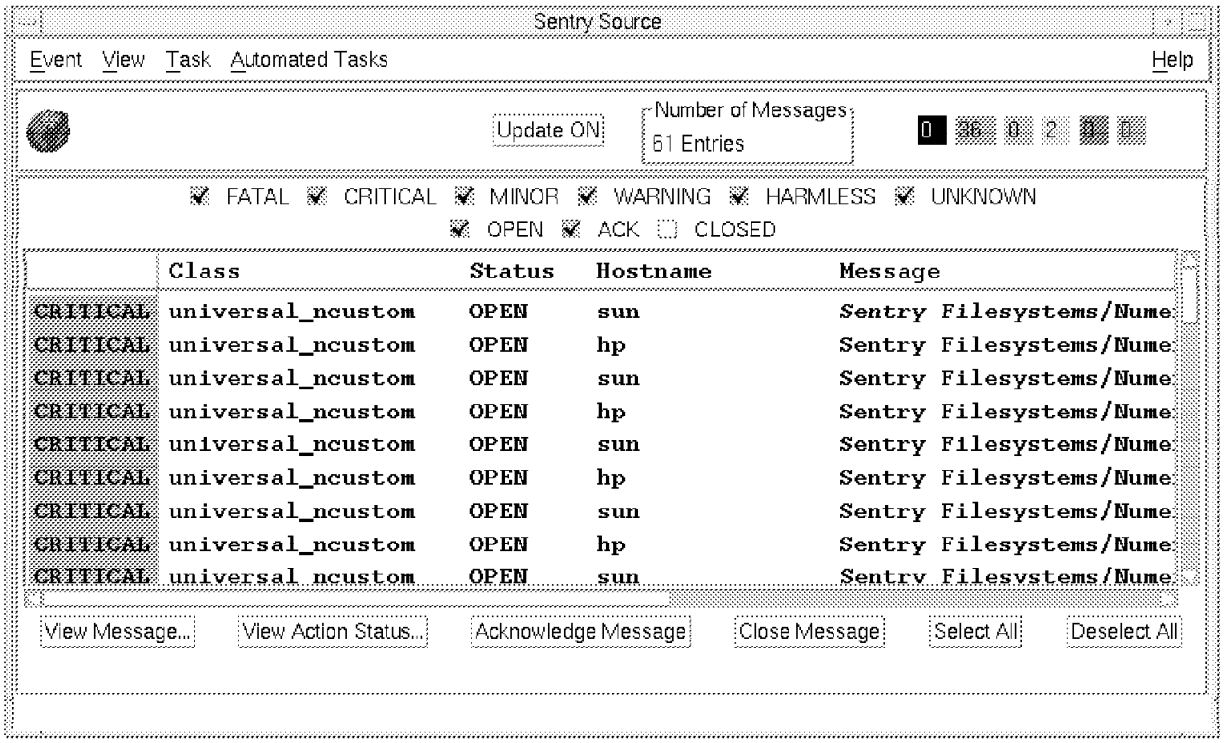



*Figure 46. TME 10 Distributed Monitoring Events Reaching the T/EC*

To see the details (slots) for a specific event, select the event with the left mouse button and then press the **View Message...** button. This will display the event slots for the event, as shown in Figure 47 on page 61.

What you will normally want to do now is, for example, add an automated action for specific events or correlate events from TME 10 Distributed Monitoring with events from other sources.

We will not show how to do this here. However, you can refer to the redbooks *TME 10 Cookbook for AIX*, SG24-4867 or *The TME 10 Deployment Cookbook: Courier and Friends*, SG24-4976 to get examples of how to do that.

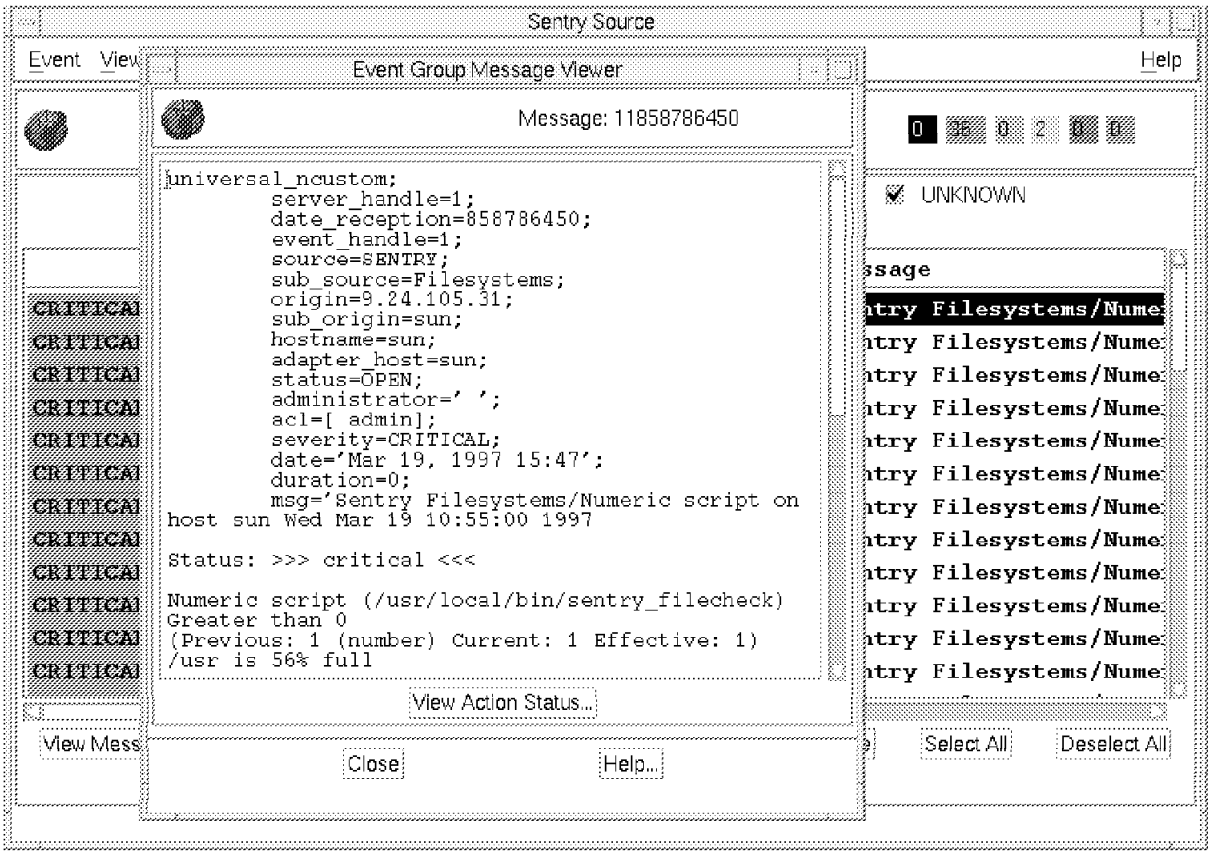



*Figure 47. Event Slots of a Received Event*

### 5.1.4 Sending Events to TME 10 NetView

If you are migrating from a Systems Monitor environment you will most likely have TME 10 NetView as your SNMP management system. One of the key features of NetView is its event display and automation facilities. In many environments TME 10 NetView is used as the focal point for receiving events.

In TME 10 NetView, all events are stored as SNMP traps. Some of the events are generated internally by NetView and some are real SNMP traps, created by an SNMP agent somewhere in the network and directed to NetView. When they arrive, the (NetView) event console displays the messages. Systems Monitor MLM, SLM and SIA all use traps to indicate problem or resolution events.

You may want to retain NetView as the focal point for your event management, even after replacing SIA and SLM with TME 10 Distributed Monitoring. Even if you intend to ultimately use the TME 10 Enterprise Console (T/EC) for this role, there may be a transition period in which you want to display Sentry events on a NetView console.

In this section we will show two techniques that allow you to do this.

### 5.1.4.1 Sending Alerts Using the NetView Event Command

The easiest way to simulate an event in TME 10 NetView (either the arrival of a trap or an internal event) is to use the event command. If NetView is on a machine that has the TME framework installed (either the TMR server or a managed node), events can be sent from TME 10 Distributed Monitoring to NetView using the **Run Program** option as a monitor response to execute the /usr/OV/bin/event command on the NetView system.

Figure 48 shows an example of a monitor definition that invokes a shell script on the system where NetView is running. The script, in turn, executes the event command.

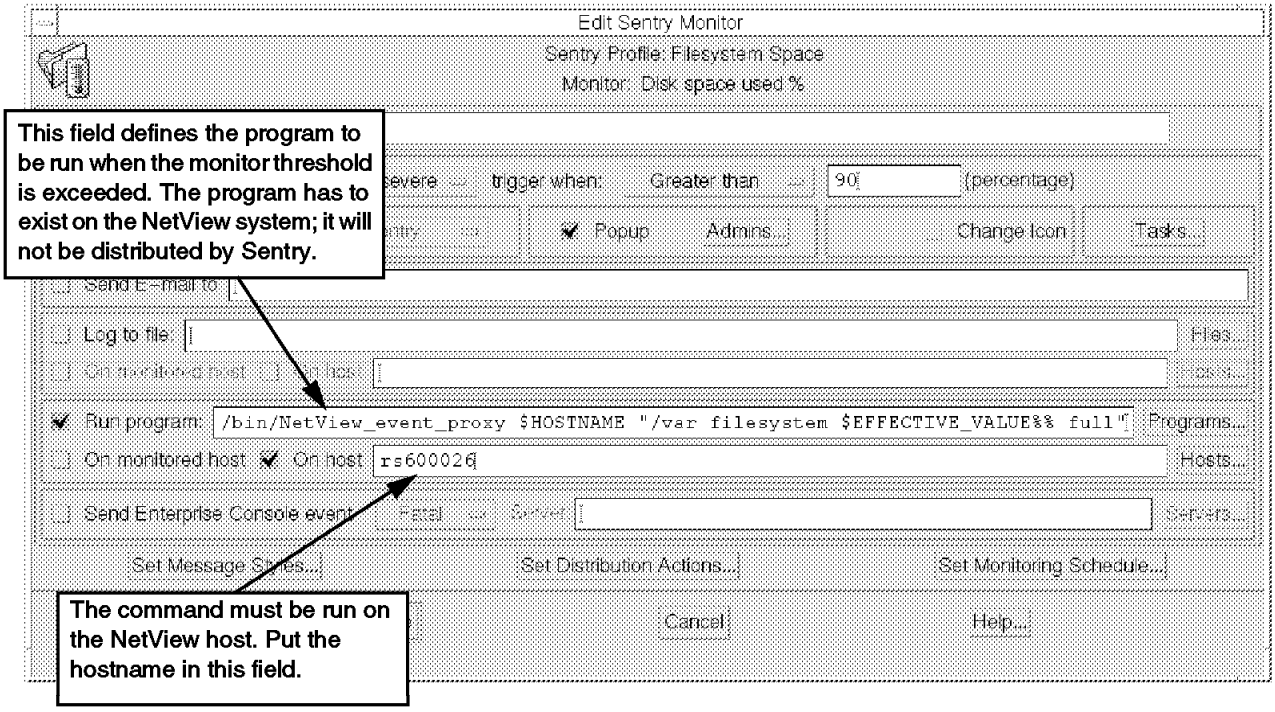



*Figure 48. An Example of Sending an Alert to NetView Using the event Command*

This example uses two Sentry Environment variables:

- $HOSTNAME - Contains the name of the local managed node
- $EFFECTIVE_VALUE - Contains the output from the monitor. In this case it is the percentage utilization of the file system.

Note also that the percent sign (%) is a special shell character, so it has to be escaped to cause it to be passed in the argument to the shell script correctly.

The script that we invoke here, NetView_event_proxy, is part of the package of samples for this book. Appendix B, “How to Get the Samples in This Book” on page 141 tells you how you can obtain it. The script is listed in Figure 49 on page 63. Note that even though the /usr/OV/bin/event command is run on the NetView host, the script substitutes the real monitored host name, extracted from the $HOSTNAME environment variable.

```
#!/bin/ksh
#!/bin/ksh
#########################################################################
# FILE:         NetView_event_proxy
#
# This script is an example from IBM redbook SG24-4936. It is made freely
# available on the understanding that it is unsupported sample code only.
#
# DESCRIPTION:
# This script can be run from a Sentry monitor to send an alert to
# TME 10 NetView.
#
# ARGUMENTS:
# (1) The hostname on which the monitor ran
# (2) The text message returned by the monitor
#
# RETURNED VALUES:
#   This script returns zero if successful, one otherwise.
#
# AUTHORS:      Graeme Naysmith IBM UK
#               Rob Macgregor   ITSO Raleigh
#########################################################################

if [[ $# < 2 ]]
then
  print "Please supply a hostname and message"
  exit 1
fi

# The event command will generate a NetView Application Event
# with the specified hostname and text inserted.
#
/usr/OV/bin/event -s A -h $1 -e AA_EV -d "$2"
```

*Figure 49. NetView_event_proxy Script*

The result of the file system monitor invoking this script is the NetView event shown in Figure 50 on page 64.

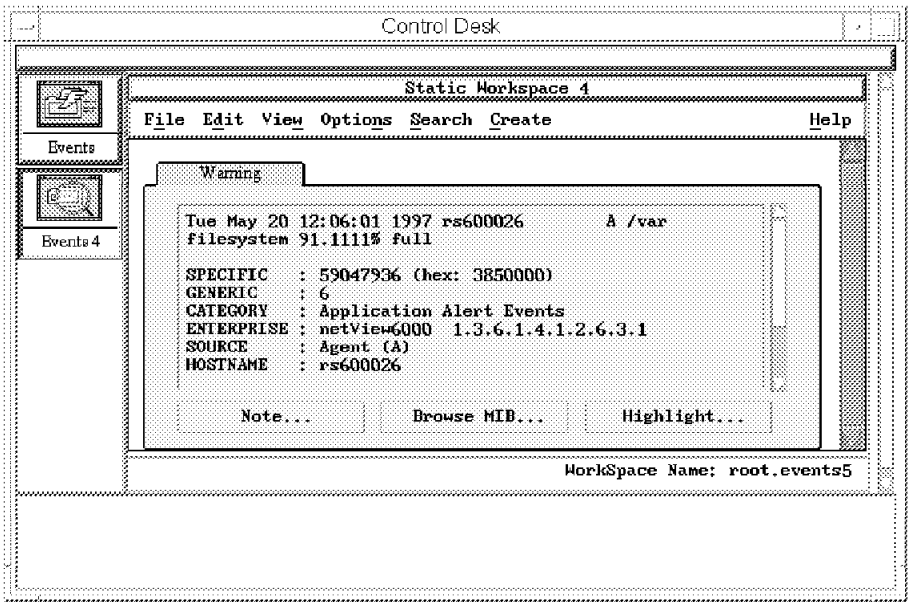



*Figure 50. NetView Event Generated from a Sentry Monitor*

### 5.1.4.2 Sending Alerts to NetView Using SNMP Traps

The example we have just described takes advantage of the TME platform capabilities for reliable, secure remote command execution. However, if the NetView machine is not on a TME node we have to use some other technique to get the alert to it. The easiest way is to send them in the format that NetView understands: SNMP traps.

NetView provides a command, snmptrap, which will generate and send a trap. However, it needs to be executed on the system where the Sentry monitor is running, not the NetView machine. It is not straightforward to copy the snmptrap executable to another system. Fortunately, Systems Monitor SIA gives us an alternative version of snmptrap, so if you are migrating from Systems Monitor you have the function you need available. It is located in /usr/lpp/smsia/original/snmptrap.

To invoke the command we need to create a TME 10 Distributed Monitoring monitor that invokes it. The command is rather complex, so the easiest thing is to place it in a shell script, such as the one shown in Figure 51 on page 65.

```
#!/bin/ksh
#########################################################################
# FILE:         SNMP_trap_proxy
#
# This script is an example from IBM redbook SG24-4936. It is made freely
# available on the understanding that it is unsupported sample code only.
#
# DESCRIPTION:
# This script can be run from a Sentry monitor to send an alert to
# TME 10 NetView in the form of an SNMP trap.
#
# ARGUMENTS:
# (1) The hostname on which the monitor ran
# (2) The alert priority (numeric, 1-7. Translated to generic trap ID)
# (3) The text message returned by the monitor
#
# RETURNED VALUES:
#   This script returns zero if successful, one otherwise.
#
# AUTHORS:      Graeme Naysmith IBM UK
#########################################################################

EVENT_DEST="rs600026"

if [[ -r /usr/local/lib/automation_flag ]]
then
  cat /usr/local/lib/automation_flag | read flag

  if [[ $flag = "automation_off" ]]
  then
    exit
  fi
fi

if [[ -r /usr/lpp/smslm/original/snmptrap ]]
then
  /usr/lpp/smslm/original/snmptrap $EVENT_DEST public 1.3.6.1.4.1.2.3.1.2.1.1.2.
1 $1 6 $2 0 1.2.3.6 octetstring "$3"
elif [[ -r /usr/lpp/smsia/original/snmptrap ]]
then
/usr/lpp/smsia/original/snmptrap $EVENT_DEST public 1.3.6.1.4.1.2.3.1.2.1.1.2.1
$1 6 $2 0 1.2.3.6 octetstring "$3"
elif [[ -r /usr/lpp/sm6000/original/snmptrap ]]
then
  /usr/lpp/sm6000/original/snmptrap $EVENT_DEST public 1.3.6.1.4.1.2.3.1.2.1.1.2
.1 $1 6 $2 0 1.2.3.6 octetstring "$3"
else
  #print "Systems Monitor snmptrap not found" > /tmp/sendtrap | tee `mail -s "se
ndtrap failure" netview@prnv << /tmp/sendtrap`
  print "Systems Monitor snmptrap not found"
fi
```

*Figure 51. SNMP_trap_proxy script*

To invoke the script from a TME 10 Distributed Monitoring monitor, specify the pathname under which the script is stored in the Run program option, exactly as we did before, but this time specify that the command is to run on the monitored host (in fact, you could specify that it runs on *any* TME host that has the snmptrap executable installed). Figure 52 on page 66 shows an example of this, in which we assume that the script to send the SNMP trap is stored in /usr/local/bin.

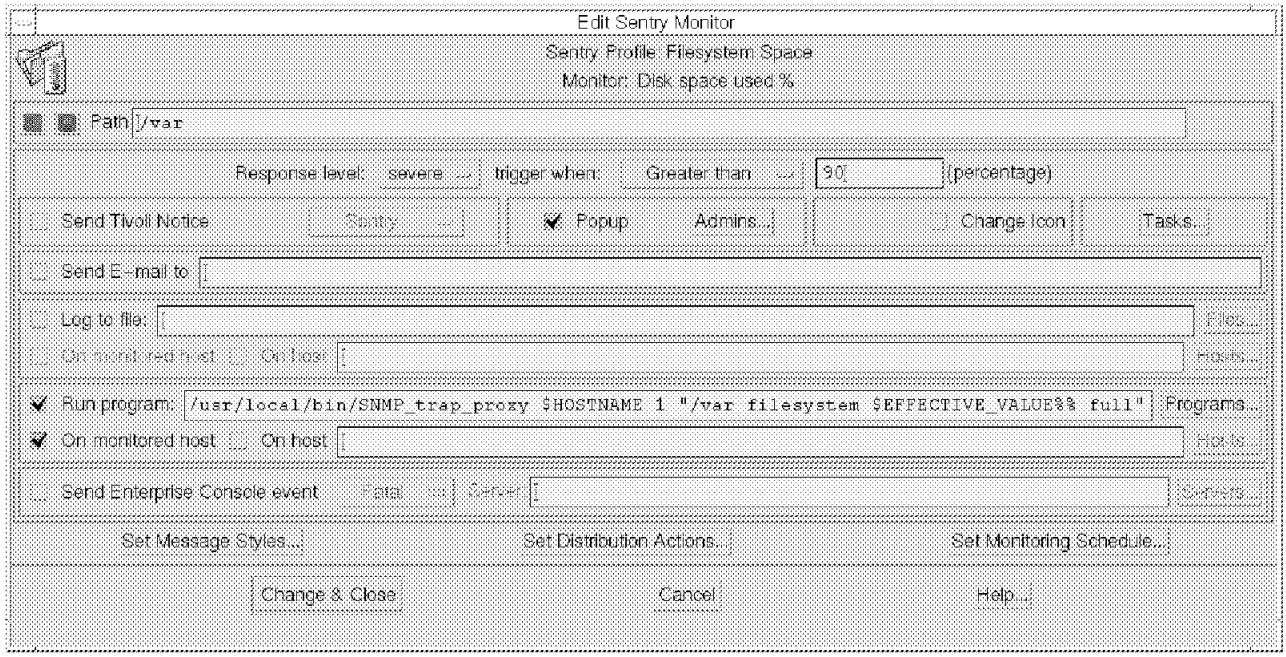



*Figure 52. Example of Using an SNMP Proxy*

After activating the monitor, SNMP events will be sent to TME 10 NetView. They will appear exactly the same as the traps generated locally on the NetView machine, as shown in Figure 50 on page 64.

## 5.2 Print Subsystem Monitoring

The redbook IBM Systems Monitor for AIX, Anatomy of a Smart Agent, SG24-4398, shows a series of examples using Systems Monitor for print subsystem (lpd) management. We do not re-create the entire example here using TME 10 Distributed Monitoring, but we do show some useful techniques.

There are two main areas to be addressed in print management:

- Checking that the queues are available and not too large
- Checking that the lpd daemon is running

### 5.2.1 Monitoring with and/or Migration to Sentry

When monitoring print queues from Systems Monitor, it was necessary to add a command to perform the checking process. TME 10 Distributed Monitoring makes life much easier by providing a number of monitors (in the Unix_Sentry monitoring collection), as follows:

**printstat** Status of print queue (up or down)

**printjobs** Number of jobs in print queue

**printjobsize** Total size of all queued print jobs

All of these monitors require a single argument, namely, the name of the print queue to be checked. This means that a separate monitor needs to be added for each queue via the GUI or CLI. As an alternative to this we created a custom monitor, which checks all the queues, and restarts any which are found to be down.

Figure 53 on page 67 shows this generic queue monitor script. It produces a simple response of 0 or 1, depending on whether all print queues are functioning or not.

```
#!/bin/ksh
#########################################################################
# FILE:         Print_queue_restart
#
# This script is an example from IBM redbook SG24-4936. It is made freely
# available on the understanding that it is unsupported sample code only.
#
# DESCRIPTION:
# This script was designed to be run by Tivoli Sentry Version 3.0
# to provide generic print queue monitoring.
# run from a universal numeric monitor with trigger set "Greater than 0"
#
# Only tested on AIX 4.2
#
# ARGUMENTS:
# none
#
# RETURNED VALUES:
#   Returns 0 if all print queues are operational, 1 otherwise
#
# AUTHOR:       Graeme Naysmith IBM UK
#########################################################################

system=`uname`

case $system in

  AIX)

  lpstat -a | awk 'BEGIN{getline}
  { if($3=="DOWN")
  {system("qadm -U" $1)}
  }'

  lpstat -a | grep DOWN > /dev/null

  if [[ $? -ne 0 ]]
  then
    print "0"
  else
    print "1"
  fi

;;

 *)
    print "# $0 Operating system not recognised"
  ;;

esac
```

*Figure 53. Print Queue Monitoring and Restart Script*

## 5.2.2 Automating Daemon Recovery

When you are thinking of monitoring a machine, often one of the first requirements is to monitor the status of processes. Both Systems Monitor and Sentry provide facilities for monitoring the state of a daemon. Although this seems like a simple thing to do, it can become complex, for example, if you need to monitor multiple related daemons, or perform some kind of automated restart. Usually there are several ways to do this.

In this section we use the print server daemon, lpd, as an example. We monitor its status, in case it dies, and show the different approaches to restarting it automatically. The restart options are as follows:

1. In the Sentry monitor, run a command such as `startsrc -s lpd` as an automated action, as shown in the following figure. The command startsrc is the AIX subsystem resource controller command to start a subsystem. Other operating systems would use different commands to perform the same function.

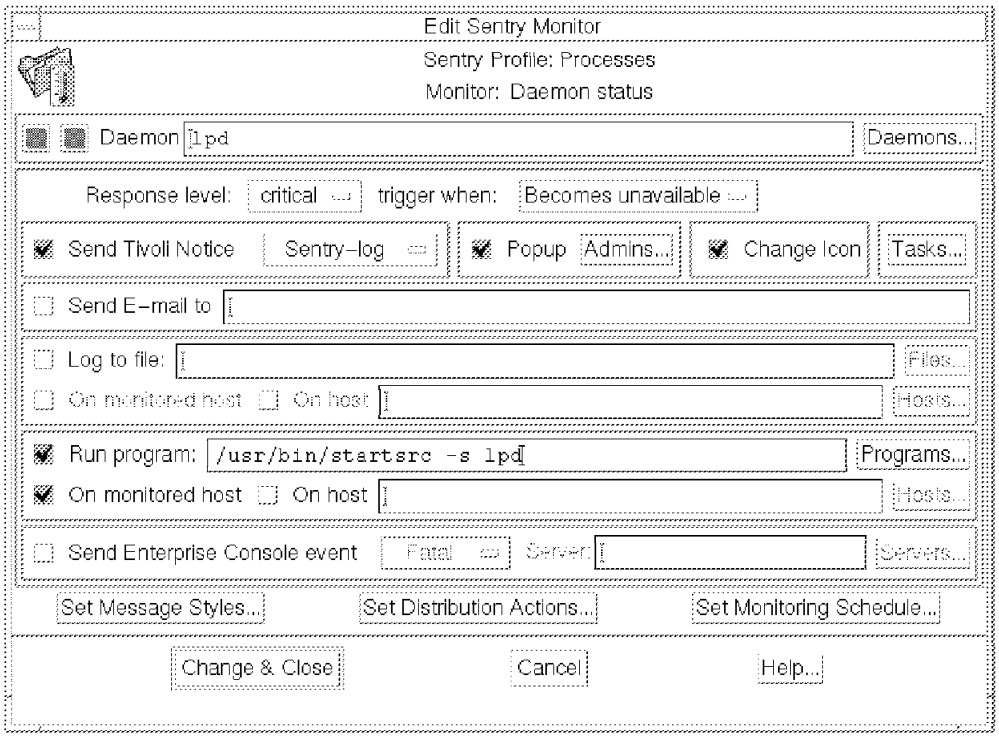



*Figure 54. Run Program startsrc*

2. In the Sentry monitor, run a TME 10 task by invoking it as a program using a command line like:

   ```
   wruntask -h rs600026 -t Startsrc -l Printers -a "-s" -a lpd
   ```

   This is very similar to running a simple command as shown in Figure 54. However, you can utilize the advantages of a TME 10 task over a standard script or command. One advantage is that the command to be executed does not have to be installed on the system. The TME task function will retrieve it and install it temporarily. Another advantage is that tasks can be made system independent, so you can use the same definition for any type of UNIX.

   Normally you would not run a task in this way, because there is a facility to run a task directly, instead of invoking it using the wruntask command. Figure 55 on page 69 illustrates this.

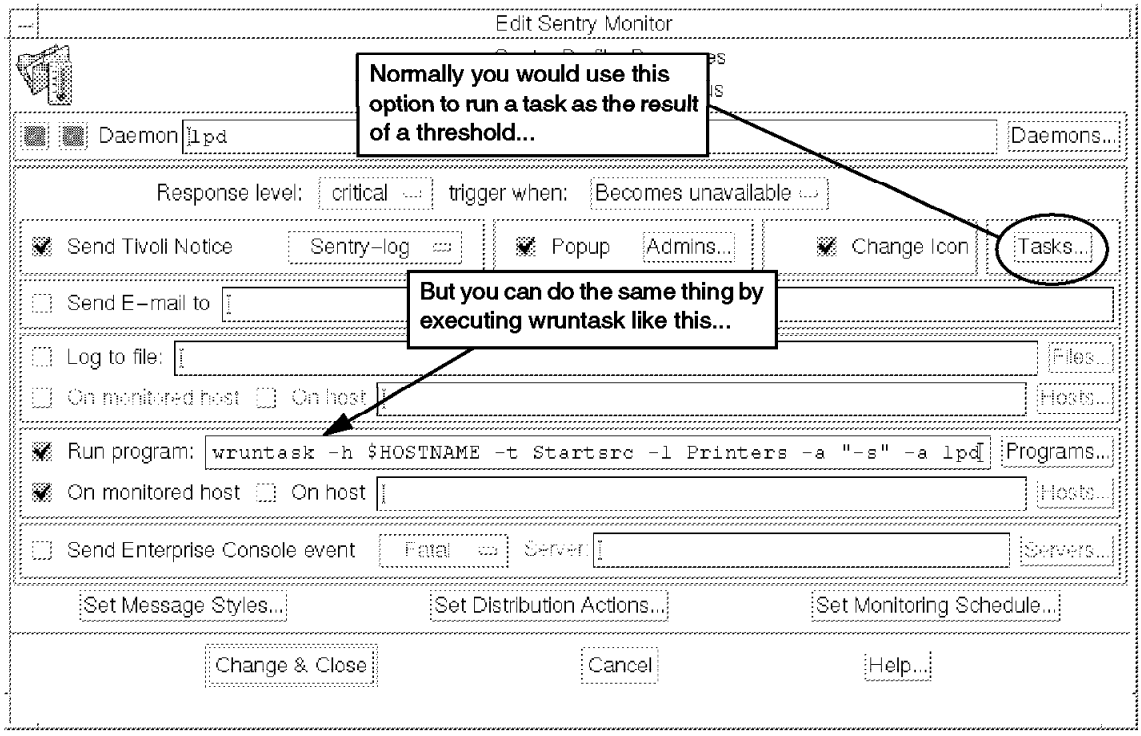



*Figure 55. Running a TME Task from a Sentry Monitor Using wruntask*

3. In the Sentry monitor, run a task with the necessary flags instead of running a program. To do this you click on **Tasks** (see Figure 55) and enter the task details, as shown in Figure 56.

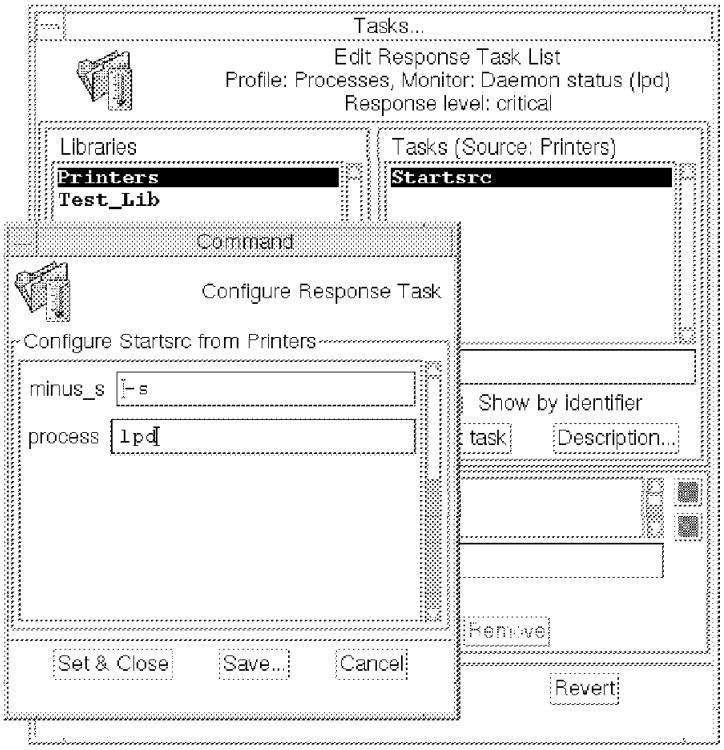



*Figure 56. Selecting a Task to Execute*

Both examples 1 and 2 invoke the same task, Startsrc, from the Printers task library. Where did this task come from? In fact, we created it ourselves using the GUI to create the initial definition and then modified it using the *Task Library Language* (TLL). We show you how we created the task library and task and how we extended its functionality in the next section.

More information about tasks and task libraries can be found in:

- *TME 10 Task Library Language Developer's Guide,* SC31-8436
- *TME 10 Framework User's Guide,* GC31-8433

You can also find another example of using TLL in *TME 10 Cookbook for AIX*, SG24-4867.

## 5.2.3 Using TME 10 Tasks with TME 10 Distributed Monitoring

Before creating it, let us review exactly what a task library is.

### 5.2.3.1 What Is a Task Library?

In TME 10, a task library is a collection of tasks. Each task represents a program that can be executed on a TME Managed Node. A task has several advantages compared to a normal program execution:

- A task must be associated with a TME Resource Role; therefore you have control over security.
- A task can be run under an arbitrary user ID on the system on which it is executed by temporarily assigning special privileges just for that task (for example, the help desk should be able to run a system backup and needs system privileges just for that task).
- The organization into task libraries and tasks establishes a common way to store automated routines. So, you don't have to worry about updating and distributing your automated action programs.
- The Task Library Language can be used to furnish a task with a simple user interface which the administrator can use to supply input parameters to the task without having to remember a complex syntax.

### 5.2.3.2 Prerequisites

We add a TME 10 Distributed Monitoring monitor that checks for daemon status, as shown in Figure 57 on page 71. Click on **Add Empty** when you have entered the details.

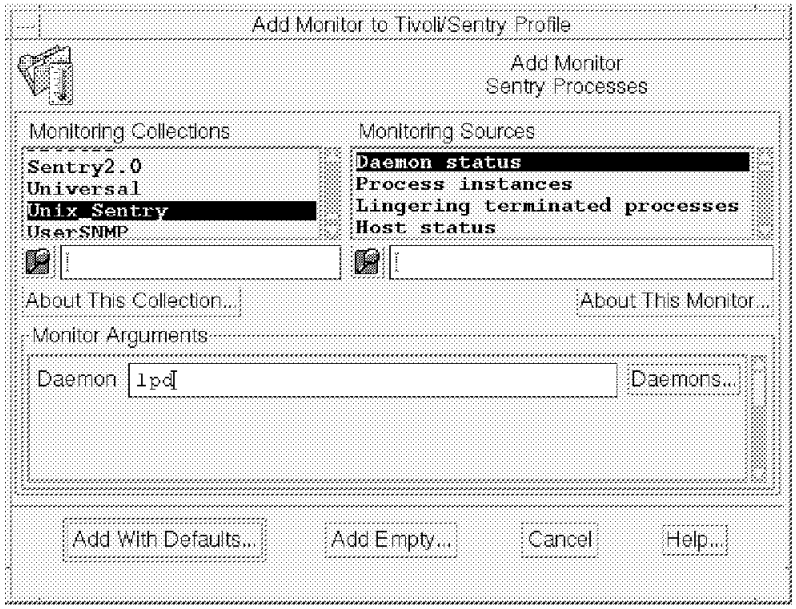



*Figure 57. Adding a Monitor to Watch the lpd Daemon*

In the details screen, select the response level that you want to be triggered when the lpd daemon fails and set it to trigger when the daemon becomes unavailable. Fill in the rest of the details as shown in Figure 55 on page 69. Click on **Set Monitoring Schedule** and define a fairly frequent interval, 3 minutes for example. Finally, save the new monitor by clicking on **Change and Close** to return to the Sentry Profile Properties screen. Next you need to worry about the user and group ID the monitor will execute under. Any ID can check for the daemon status, but it needs system privileges to be able to restart it. Click on the monitor and then select **Edit** followed by **Set User and Group ID** from the menu bar (see Figure 58).

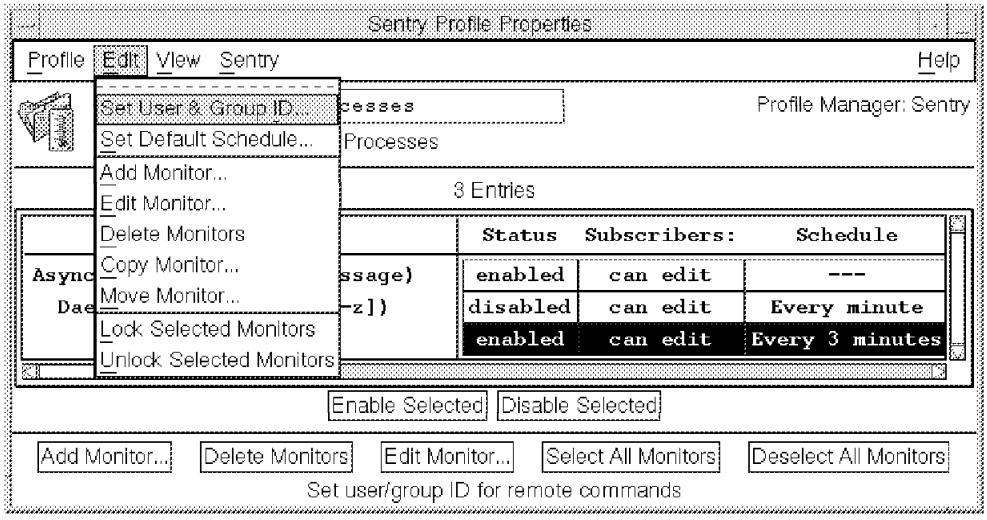



*Figure 58. Defining User and Group ID for a Monitor*

By default, monitors and the automated actions they invoke are run as user nobody and group nobody. However, this is not enough to restart the lpd daemon. You have to be at least a member of the system group.

We created an AIX user ID called xxx, which has the system group in its group set. We can then set the user ID and group ID for the TME 10 Distributed Monitoring monitor as shown in Figure 59 on page 72.

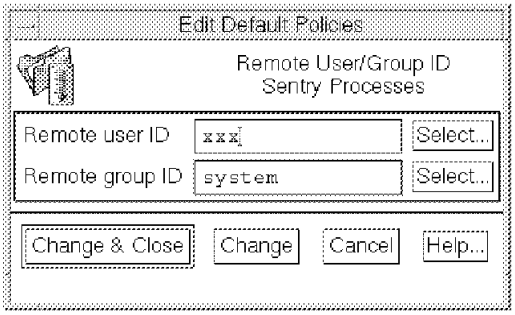



*Figure 59. Setting IDs*

### 5.2.3.3 Setting Up a Basic Task

To deal with tasks, you first have to create a task library. You can do this by selecting **Create** from the menu bar in the Policy Region window and then selecting **TaskLibrary...** from the pull-down menu. Make sure that TaskLibrary is a managed resource of the policy region where you want to create the new task library.

In our example, we have created a task library called Printers. Within the task library you can then create a task, by selecting the task library icon with the right mouse button as shown in Figure 60.

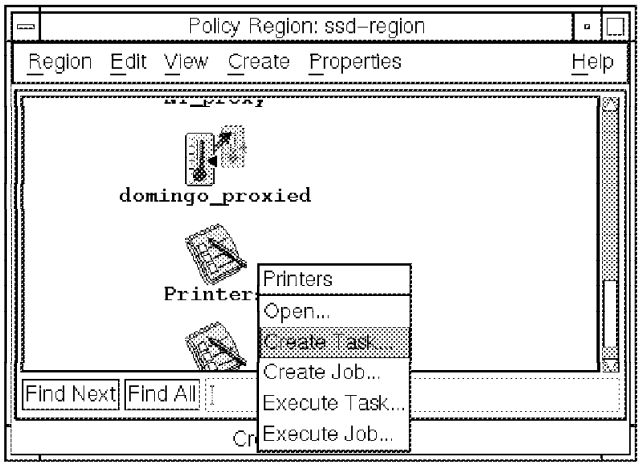



*Figure 60. Creating a Task*

The screen shown in Figure 61 on page 73 will appear. Enter the task details as indicated.

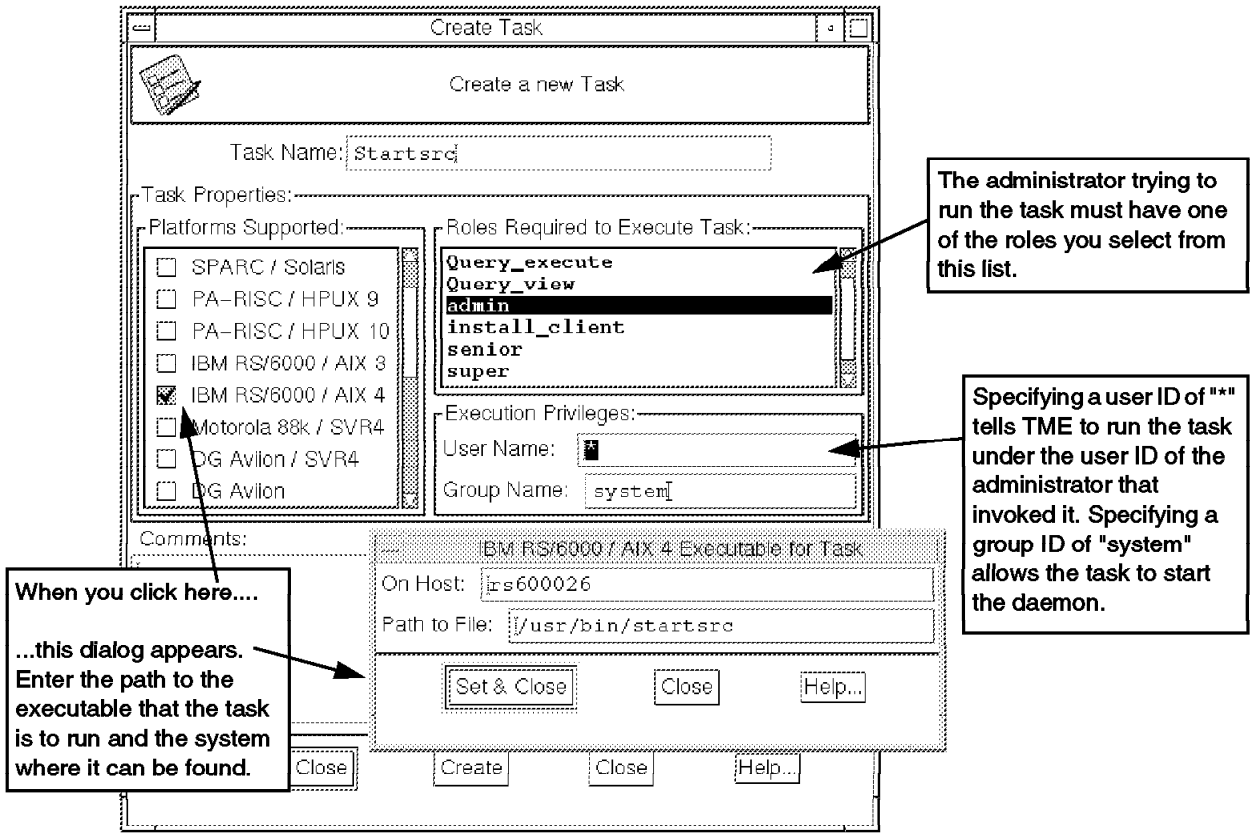



*Figure 61. Editing a Task*

**Note**

Notice that you cannot specify any arguments to pass to the command in this dialog.

Click on **Set & Close** and then **Create & Close** to finish creating the new task. This will add a new icon for the task in your task library.

Once you have finished these steps, you can test your task by double clicking it and selecting a host (**Available Task Endpoint**) to run it on. During testing you may want to select **Display on Desktop** and **Save to File** to watch the task output. Since our example deals with the startsrc command, execution of the task will *not* be successful. The startsrc command requires some flags, which our simple task definition does not allow us to provide. We will add these flags in the next section.

### 5.2.3.4 Extending the Basic Task

Once you have created the Printers task library and the Startsrc task you are ready to extend its functionality by passing it the two arguments it needs.

From a command line, enter:

```
wtll -F /tmp/Printers.tar -l Printers
```

This will extract the task library definition into a tar archive in a file called /tmp/Printers.tar. Enter the following commands to unpack the archive:

```
cd /tmp
tar -xvf /tmp/Printers.tar
```

You will get two files, named 0.aix4-r1 and tll. We are interested in the tll file. Figure 62 shows the file, with our additions highlighted.

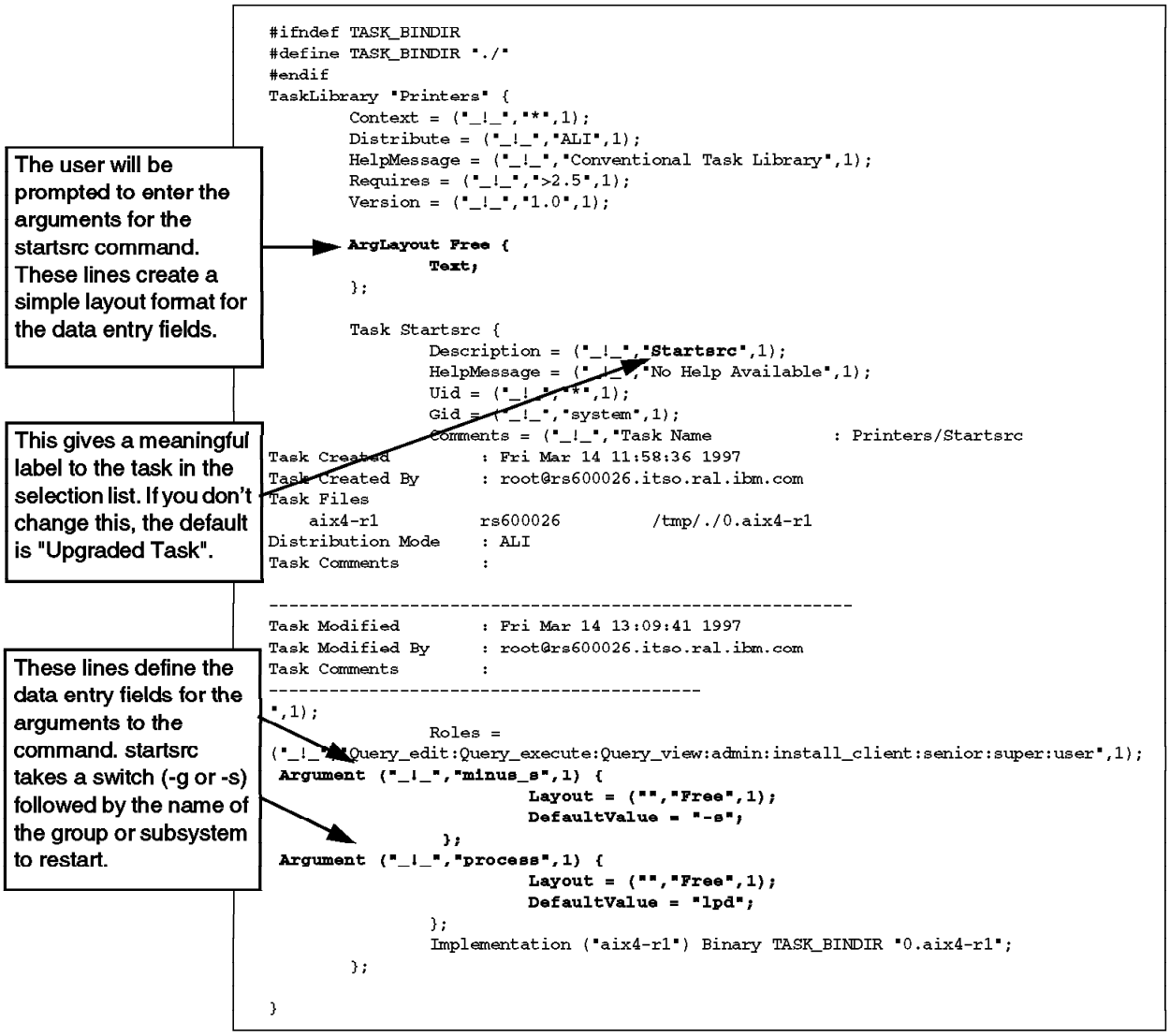



*Figure 62. tll File with Changes*

While looking at this Task Library Language definition, we recommend that you refer back to Figure 56 on page 69 to see the effect of the definitions we have added.

**Note**

We also could have written the source code shown above entirely from scratch. However, creating it using the graphical user interface and then exporting it to the Task Library Language definition saves some work.

After making the changes to the tll file, we can compile it by typing:

```
wtll -r -P /usr/ccs/lib/cpp -p ssd-region /tmp/tll
```

**Note**

This command uses the C preprocessor, cpp. Problems will arise if you do not have a working version of this. You may need to have a license key installed. In fact, we found some problems using the version of cpp that is shipped with C Set++ and we used a copy from an AIX V3R2 system instead.

Once the task library has compiled successfully, you can use the new task. Define a monitor as usual and click on **Tasks**. You will see a screen like the one shown in Figure 56 on page 69. In the left field the available task libraries are displayed, if you double click on **Printers** the available tasks appear in the right field. Make sure that you have selected the field **Show by Identifier** to see the names of the tasks. Otherwise you will only see the name ″Upgraded Task″. If you double click on the task that you want to run, a new window pops up and in our example asks you for the arguments `minus_s` and `process` that we have defined in the tll file.

### 5.2.4 Comparing the Different Approaches

All of our monitors look for the status of the lpd daemon and try to restart it if it is down. Look at Figure 63 on page 76 which is an extract of a wlsmon command. The important differing lines are highlighted:

**monitor 0** run program(/usr/sbin/startsrc -s lpd)

**monitor 1** run program(wruntask -h $HOSTNAME -t Startsrc -l Printers -a ″-s″ -a lpd)

**monitor 2** tasks:Printers:Startsrc(-s, lpd)

The first example (monitor 0) implements a very simple solution for restarting the lpd daemon. It just runs the program /usr/sbin/startsrc with some flags. The disadvantage here is that there is no easy way to run different programs on different platforms. The advantage is that you don’t have to deal with the task library and task definition.

The implementation in monitor 1 is a mixture between running a program and working with tasks. It is easy to run different programs on different platforms. A disadvantage is the rather cryptic command line. A person who is new to UNIX could get confused with, for example, the -a syntax for the parameters.

The last implementation, monitor 2, is the ″cleanest″ way to restart lpd. The task corresponding to the type of system where the monitor is run is selected automatically and you get a window prompting you to enter the necessary parameters. On the other hand, you have to learn how to create and change tasks using the Task Library Language.

```
0  Monitor: Daemon status(lpd)
  Timing:Every 5 minutes
  disabled
  Responses:
    [critical]
    when probe result  == down
      popup(Root_ssd-region),notify(Sentry),run program(/usr/sbin/startsrc -s lpd)
    [severe]
    [warning]
    [normal]
    [always]
1  Monitor: Daemon status(lpd)
  Timing:Every minute
  Responses:
    [critical]
    when probe result  -> down
      popup(Root_ssd-region),notify(Sentry-log), \
      run program(wruntask -h $HOSTNAME -t Startsrc -l Printers -a "-s" -a lpd)
    [severe]
    [warning]
    [normal]
    [always]
2  Monitor: Daemon status(lpd)
  Timing:Every 30 minutes
  Responses:
    [critical]
    when probe result  -> down
      tasks:
          Printers:Startsrc(-s, lpd)

    [severe]
    [warning]
    [normal]
    [always]
```

*Figure 63. Extract of wlsmon Output*

## 5.3 Monitoring CPU Utilization

One of the monitoring tools delivered by Systems Monitor SIA is a MIB table of CPU utilization. You can, for example, retrieve the MIB variable .1.3.6.1.4.1.2.6.12.2.9.1.3.1.6 and get the average percentage of time that the CPU was in idle mode over the interval. CPU utilization is a notoriously difficult thing to measure in a meaningful way. For example, it is normal for the CPU to hit 100% utilization for short periods of time, so you want to be able to monitor over a long enough period to be sure that the utilization is really consistently high.

There is also some debate about how useful it is to set thresholds on CPU usage. Most operating systems are designed to balance load so that a process that is using a lot of system resource does not impact the performance of other processes. This means that it is quite possible for a system to always be 100% utilized, yet the users will not feel any ill effects. On the other hand, if the utilization suddenly leaps to a high value it may be an indication of a looping application, or some other bug that needs investigation.

Unfortunately, there are no standard CPU monitors delivered in TME 10 Distributed Monitoring for the UNIX platforms. We will show a technique for implementing such a monitor. We apply this in two ways:

1. Using two shell scripts invoked by the Numeric Script monitor (from the Universal monitoring collection)

2. Implementation of a new monitoring collection using the Monitoring Collection Specification Language (MCSL)

As with all the examples in this book, these examples are available in the sample package (see Appendix B, “How to Get the Samples in This Book” on page 141).

### 5.3.1 How the CPU Monitor Examples Work

We want the monitors to have the following attributes:

- They should avoid reporting single instantaneous readings because the erratic nature of CPU utilization means such figures may be abnormally high or low.
- They should permit monitoring over multiple sampling intervals, so that utilization trends can be determined (for example, you may not mind if a server is heavily loaded for a 5-minute period, but you are concerned if it stays that way for an hour).
- They should allow us to break the system utilization into its components, that is: user, system, idle and wait.

The solution uses the vmstat command to extract utilization information. It comprises two related Sentry monitors. The first of these invokes vmstat and returns a ″current″ utilization. In addition to returning the value it stores it in a log file. The second monitor runs less frequently. It analyses the data in the log file and returns a mean value, calculated over a given number of monitoring intervals. Figure 64 illustrates this.

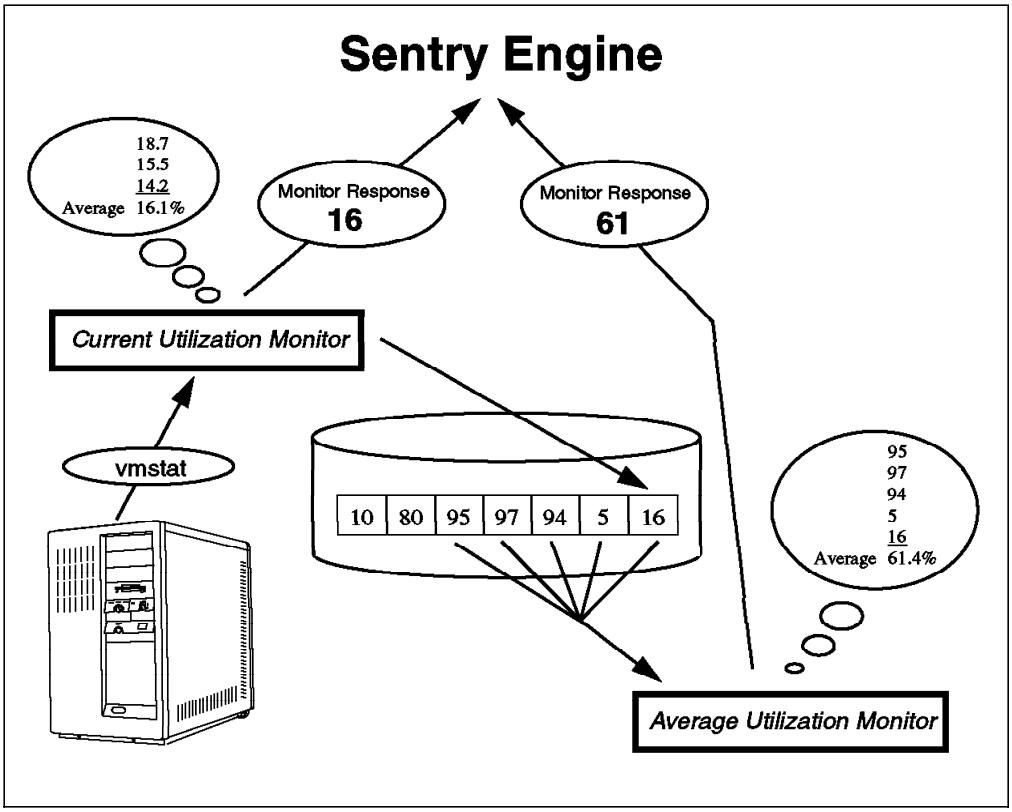



*Figure 64. Operation of the CPU Monitors*

To understand what the current utilization monitor is doing, let's have a look at the output of vmstat. The command takes two arguments, which specify the interval between probes in seconds and the number of times to repeat the probe. So if you enter vmstat 2 4 you will get four sets of utilization figures with two seconds between each. The output of the command is shown in Figure 65.

```
rs600026:/ > vmstat 2 4
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0 22984  9690   0   0   0   0    1   0 131  621  69  2  3 93  2
 0  0 22984  9689   0   0   0   0    0   0 165  173  89  0  2 98  0
 0  0 22984  9688   0   0   0   0    0   0 171  343  98  2  2 94  1
 0  0 22984  9688   0   0   0   0    0   0 120   84  46  0  0 99  0
```

*Figure 65. Output from vmstat Command*

In simple terms the work the CPU is doing can be divided into three parts: Do nothing (id), work for the system (sy) or work for the user (us). If you add these three values, the sum is always about 100% (sometimes not exactly 100% because of rounding). The vmstat command returns the percentage value of the three components and is a good indicator of how busy the CPU is at a given time.

The first output line after the header of the vmstat command is always the average value since the last system boot. Since we are interested in the current load of the CPU, we can simply ignore the first output line.

In the example shown, the CPU is not at all busy. At the first time stamp, it doesn't do anything for the user, works 2% for the system and is 98% idle. At the second time stamp it does 2% user, 2% system and is 94% idle. At the last time stamp it is idle at 99%. So we can say, over the last three time stamps the CPU was working (0+2+0) / 3 = 0.66% for the user, (2+2+0) / 3 = 1.33% for the system and was idle (98+94+99) / 3 = 97% of the time.

This is the mathematical operation that will be done by our new current utilizaion monitor. The basic idea is to have a short look (a few seconds) at the CPU, compute its average load, report it and also write the result to a file.

For the longer-term figures we will run a second monitor at a less frequent interval. So, for example, if we repeat the current utilization monitor every two minutes, the average utilization monitor may run every fifteen minutes. It performs much the same function as the other monitor, except that it works with figures from the log file instead of vmstat.

### 5.3.2 Monitoring CPU with TME 10 Distributed Monitoring Using Shell Scripts

We wrote two little shell scripts to implement the current and average CPU monitoring functions discussed above. The scripts are called cpuload.sh and cpustat.sh, respectively. They are listed in Figure 66 on page 79 and Figure 68 on page 81.

```
#!/bin/ksh
################################################################################
# FILE:         cpuload.sh
#
# This script is an example from IBM redbook SG24-4936. It is made freely
# available on the understanding that it is unsupported sample code only.
#
# DESCRIPTION:
# This script can be run using the numeric script monitor to store
# CPU load information into a file. A second monitor, running script
# cpustat.sh reads the file and reports average utilization.
#
# ARGUMENTS:
# (1) The element of CPU load to measure
#     -user         Percent CPU utilization for user processing
#     -system       Percent CPU utilization for system processing
#     -idle         Percent CPU idle
#     -system_user  Percent CPU utilization for system and user (%busy)
#
# RETURNED VALUES:
#   This script returns the current percentage utilization if
#   successful. -1 otherwise
#
# AUTHORS:      Andreas Kuffer  IBM Germany
#               Rob Macgregor   ITSO Raleigh
################################################################################
limit=30
CMD=$1
SEC=2
COUNT=4
F=6
wrapfile="/tmp/bc.wrap$CMD"

# Trim the file if it has too many entries in it
if [ -f $wrapfile ]
then
  actual=`cat $infile | wc -l`
  if [ $actual -gt $limit ]
    then
    tail -$limit $infile >> /tmp/value.tmp
    mv $infile $infile.old
    mv /tmp/value.tmp $infile
  fi
fi

# Execute the vmstat command and extract the appropriate value
case $CMD in
  "-user" )        var=`vmstat $SEC $COUNT | awk '{ printf " + %s",$14 }'`;;
  "-system" )      var=`vmstat $SEC $COUNT | awk '{ printf " + %s",$15 }'`;;
  "-idle" )        var=`vmstat $SEC $COUNT | awk '{ printf " + %s",$16 }'`;;
  "-system_user" ) var=`vmstat $SEC $COUNT | awk '{ printf " + %s + %s",$14,$15
}'`
                   F=10;;
  * ) echo "don't know how to handle $CMD"
exit -1;;
esac
# Remove the data from the header and first result line
bccmd=`echo $var |cut -d+ -f$F-`

# Calculate the mean value
bc << EOF | tee -a $wrapfile
ibase = 10
obase = 10
scale = 2
n=$COUNT-1
s = $bccmd
s / n
quit
EOF
```

Note that the monitor writes its output to a file that includes the monitor type in its name. So for example if you specify "-idle" as the argument, the file will be /tmp/bc.wrap-idle.
This means that you should only have one monitor active that invokes cpuload.sh with any given argument .

The script uses the UNIX line command calculator, bc, to derive the average of the last three figures from vmstat.

*Figure 66. cpuload.sh Script*

The cpuload.sh script expects one parameter that determines which CPU load to monitor: user mode, system mode, idle time or a combination of system and user mode (which is the total percentage of time the CPU is busy).

To launch the script from within a Sentry monitor, we use the Numeric script monitor from the Universal Monitoring Collection as shown in Figure 67 on page 80.

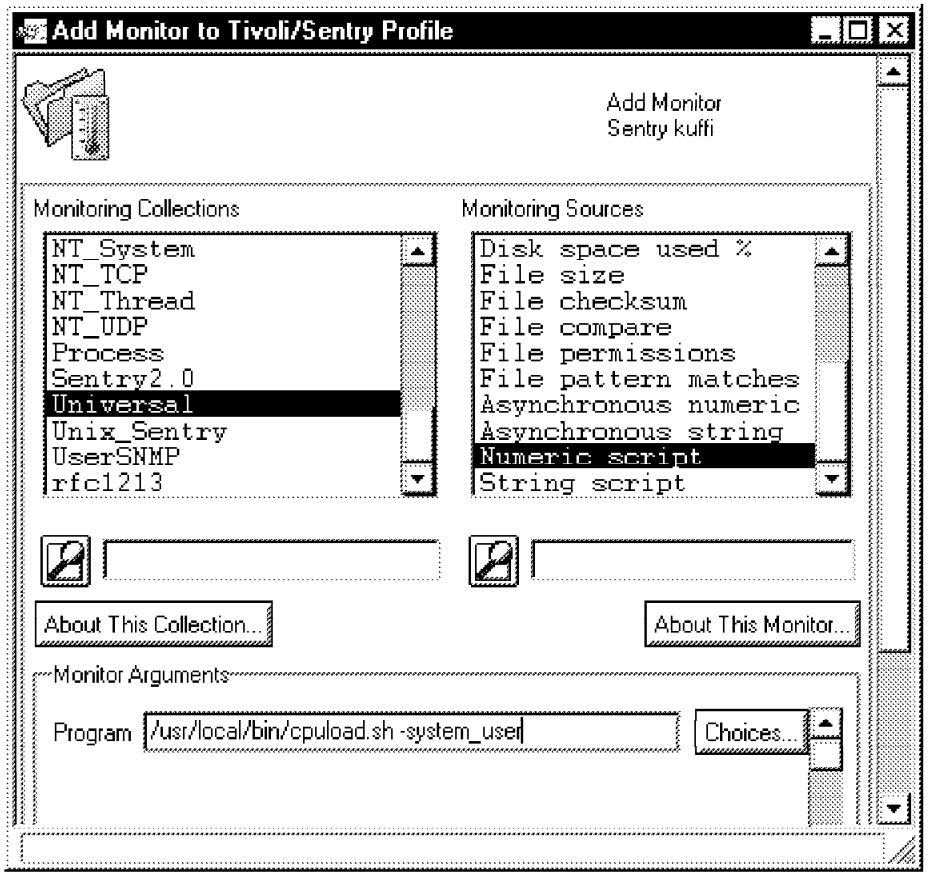



*Figure 67. Running cpuload.sh From a Numeric Script*

If you want to monitor current utilization using this script you can set up the response levels and actions as for any Sentry monitor. Alternatively if you just want to set up cpuload.sh to feed the longer-term monitoring offered by cpustat.sh you only need to set the monitoring interval and allow the response levels to all default to ″never″.

The cpustat.sh script uses the file created by cpuload.sh. It reads the CPU load data collected by cpuload.sh and builds the average over a certain number of probe intervals. The number of probes is determined by the second argument passed to the script. The cpustat.sh script is listed in Figure 68 on page 81.

```
#!/bin/ksh
#########################################################################
# FILE:         cpustat.sh
#
# This script is an example from IBM redbook SG24-4936. It is made freely
# available on the understanding that it is unsupported sample code only.
#
# DESCRIPTION:
# This script can be run using the numeric script monitor to read
# CPU load information from a file, previously written by cpuload.sh.
# It returns the mean load value over the most recent N samples
#
# ARGUMENTS:
# (1) The element of CPU load to measure
#     -user         Percent CPU utilization for user processing
#     -system       Percent CPU utilization for system processing
#     -idle         Percent CPU idle
#     -system_user  Percent CPU utilization for system and user (%busy)
# (2) The number of intervals to calculate the average over
#
# RETURNED VALUES:
#   This script returns the average utilization as an integer percentage
#
# AUTHORS:      Andreas Kuffer  IBM Germany
#               Rob Macgregor   ITSO-Raleigh
#########################################################################
utiltype=$1
count=$2
infile="/tmp/bc.wrap$utiltype"

# Calculate the average of the last $count entries
bc << EOF
ibase = 10
obase = 10
scale = 0
n=$count
s = 0`tail -$count $infile | awk '{ printf " + %s",$1 }'`
s / n
quit
EOF
```

*Figure 68. cpustat.sh Script*

The cpustat.sh script is invoked in a monitor in exactly the same way as cpuload.sh (see Figure 67 on page 80) except that it has a second argument, the number of intervals over which to summarize the results.

The relationship between the two scripts means that you can have multiple cpustat.sh monitors running at once, working with data collected by one cpuload.sh monitor. For example:

- Run cpuload.sh -system_user at 2-minute intervals to record CPU busy data.
- Run cpustat.sh -system_user 5 at 15-minute intervals to report the 10-minute average CPU utilization.
- Run cpustat.sh -system_user 30 at one hour intervals to report hourly averages.

### 5.3.3 Monitoring UNIX CPU Utilization Using MCSL

The two scripts described above meet our requirements for UNIX CPU monitoring, but it would be better if they were implemented as part of a normal monitoring collection, rather than having to distribute scripts and employ the numeric script monitor to drive them. In this section we show how to turn the scripts into a new monitoring collection, using the Monitoring Capability

Specification Language (MCSL) to create the definition. MCSL is comparable to the TME 10 Task Library Language described in 5.2.3.1, “What Is a Task Library?” on page 70.

While this approach requires more effort, it has many advantages over using generic script monitors. For example:

- The scripts do not have to be distributed separately from the monitor definition.
- An administrator does not need to know the arcane command line syntax.
- A single monitor can be made to support multiple system types.
- The monitor is portable and can easily be installed on another TME system.

### 5.3.3.1 Before You Start

To create and compile the MCSL definitions you need the following environment in place:

- Tivoli Management Platform, Version 2.0.2 or later
- Tivoli/Sentry 2.0.2 or later
- A C-language preprocessor

  On AIX, make sure you have a valid license key or copy the cpp command from an AIX 3.2.5 system, which will work without a license key.
- Access to at least the following files:

  ```
  Sentry2_0.dsl
  operators.csl
  choicelists.csl
  formats.csl
  ```

  These are include files that define messages and data structures. They are included with the TME 10 Integration Toolkit, which is available to Tivoli partner vendors. We only had the Integration Toolkit on the NT platform, so we copied the files from C:plusplusincludegenerictivoli to our working directory on AIX. We had to remove the first three or four comment lines of the include files because the AIX C pre-processor did not know how to deal with them.

### 5.3.3.2 Overview of Creating a Monitoring Collection

There are four steps to creating a monitoring collection with MCSL.

1. Create the message catalog.

   MCSL refers to messages using a label, instead of having them coded directly into the monitor definition. This allows you to translate an application into different languages and locales without changing the base function.

2. Create the monitor definition file.

   This is the heart of the process. This file defines the name of the monitoring collection and the monitors in it. It specifies the arguments that have to be passed to the monitor scripts, including the syntax of each argument. GUI elements, such as selection lists and data entry fields, are also defined. Finally, the file defines the monitor scripts themselves, either directly within the file or by reference to a separate shell script.

3. Compile the monitor definition file.

This step uses the mcsl command, which invokes the C pre-processor to parse the monitor definition and message catalog and then generates a collection definition file from them.

4. Install the new monitoring collection.

   This step again uses the mcsl command to install the collection into the running TME nodes.

We will now show how we performed each of these steps to create the UNIX CPU monitoring collection example.

#### 5.3.3.3 Creating the Message Catalog

The message catalog is defined in a .msg file (cpu.msg, in our example). Each entry in the file contains two lines:

```
$ key=keyname
n message_body
```

The keyname field is a unique key that is used to access the message from the monitor definition. The n field is a unique message number and the message_body field contains the text that is to be displayed. You need to have an entry for each text item that will appear on the user interface, such as collection and monitor descriptions, field descriptions for command arguments, selection list entries and help text.

Figure 69 on page 84 shows the cpu.msg file that we created for our example:

```
$ key=Busy_Descr
1 Percent used
$ key=Busy_Help
2 This Monitor returns the current CPU resource utilization as a percentage. It
has three options which divide utilization into user, system or idle components,
plus a fourth option which combines system and user components (effectively - CPU busy).
$ key=Busy_val_Descr
3 Percent
$ key=Resources
4 Resource
$ key=cpuload_val_Descr
5 CPU Status
$ key=cpuload_Descr
6 Current CPU utilization
$ key=cpu_choice_idle
7 CPU Idle
$ key=cpu_choice_user
8 CPU User
$ key=cpu_choice_system
9 CPU System
$ key=cpu_choice_system_user
10 CPU System plus User
$ key=ButtonLabelChoice
11 Choices
$ key=cpu_list
12 Options
$ key=generic_help
13 The CPU collection shows statistics for the CPU usr-, system- or idle-percentage
$ key=cpustat_Descr
14 CPU utilization over time
$ key=cpustat_val_Descr
15 Percent
$ key=cpustat_Busy_Help
16 This Monitor uses the output of the Current CPU utilization \
monitor and builds the average over an adjustable number of timeframes
$ key=cpustat_argument
17 Number of Timeframes
$ key=cpustat_list
18 Options
```

*Figure 69. cpu.msg File*

After you have created the cpu.msg file, you have to translate the file into a format that can be included in the monitor definition. You use the gencmsg command to do that:

```
gencmsg cpu.msg
```

Three files will be created: cpu.c, cpu.h and cpu.dsl. The cpu.dsl file will be included in the cpu.csl file as shown in the next step.

### 5.3.3.4 Creating the Monitor Definition File

The monitor definition is in a file of type .csl (we called ours cpu.csl). Figure 70 on page 85 and Figure 71 on page 86 show the contents of cpu.csl, with notes about each section.

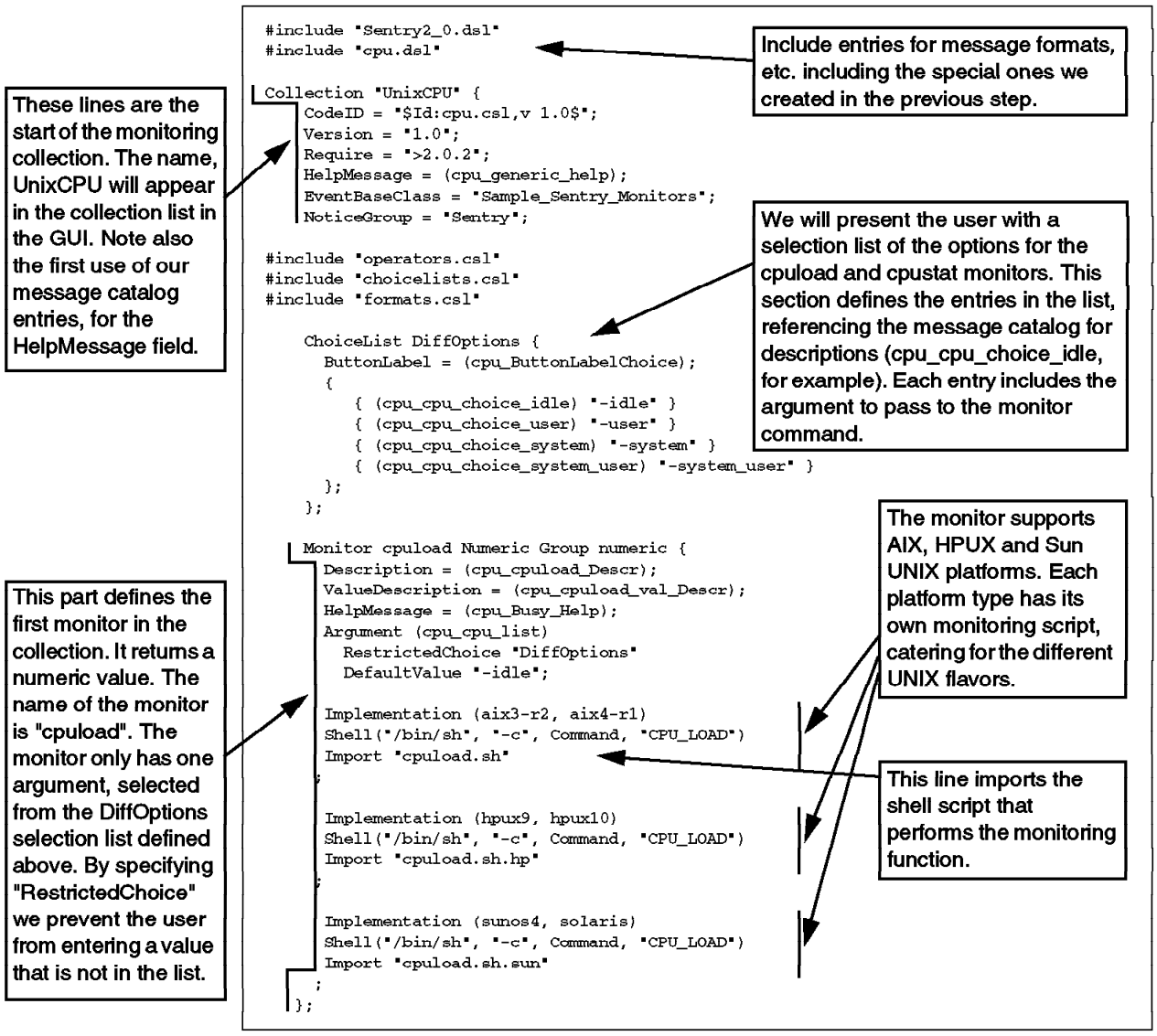



*Figure 70. Monitor Definiton File, cpu.csl (Part 1 of 2)*

Note that every time we refer to a message created in the cpu.msg message catalog file, we use the keyname preceded by the filename of the catalog and an underscore, that is, ″cpu_″ (for example, cpu_ButtonLabel_Choice in the selection list definition).

The keyword *Numeric* on the monitor definition specifies that this monitor is going to have a numeric return value. Alternatively we could have specified String or Async. The text *Group* on the same entry is a keyword and finally *numeric* is a reference to a definition in the operators.csl include file. This controls which trigger options you have for the monitor (see Figure 76 on page 89). There are two alternatives to numeric, either *string* or *available*. Specify string for triggers such as ″Equal to″, ″Matches″ and ″Changes to″. Specify available for triggers such as ″Is up/available″ and ″Becomes available″.

The cpuload monitor has three different implementations defined, for AIX, HPUX and for Sun. Each of them invokes a slightly different version of the cpuload.sh shell script that we described above (see 5.3.2, “Monitoring CPU with TME 10 Distributed Monitoring Using Shell Scripts” on page 78). The differences are

caused because the three different flavors of UNIX display the vmstat results in slightly different ways. If you want to put the complete definition into one file, it is possible to imbed the script lines in the .csl file, instead of using the Import keyword. If you do this, each line of the script must be prefixed with a period (″.″).

Now we can complete the monitoring collection definition by adding the second monitor, called cpustat, which takes the output file produced by the first monitor and builds an average over a variable number of time frames (see Figure 71).

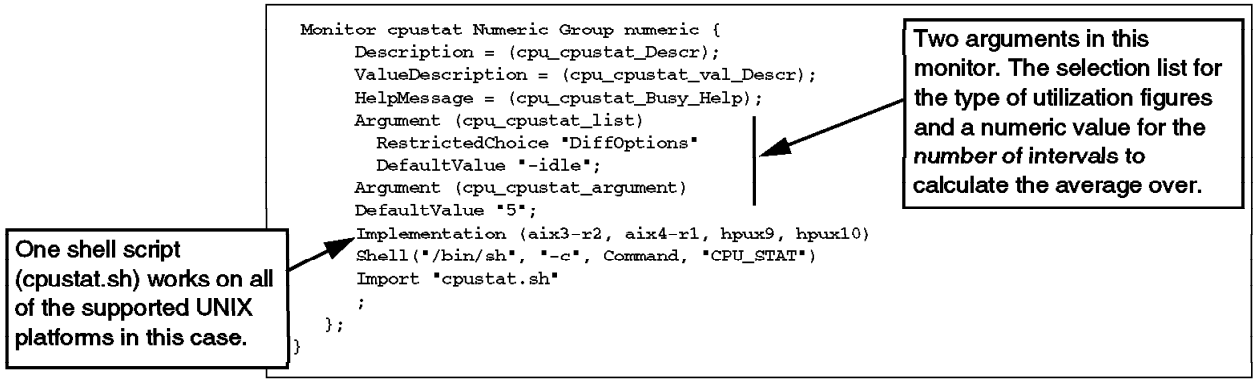



*Figure 71. Monitor Definiton File, cpu.csl (Part 2 of 2))*

### 5.3.3.5 Compiling the Monitoring Collection

Now we are ready to compile our cpu.csl file. Use the following command to do this:

```
mcsl -P /usr/lib/ccs/cpp -x cpu.col cpu.csl -lang-c++ -nostdinc -undef
```

Where:

**-P** specifies the path to the C pre-processor to be used

**-x** specifies the output file and the source .csl file

The result of this is a binary file, cpu.col, which contains the complete monitor definition, including the implementation scripts for the supported platforms.

### 5.3.3.6 Loading the Monitoring Collection

Once you have your cpu.col file, you are ready to load the collection. Use the following command to load it into TME 10 Distributed Monitoring:

```
mcsl -i -R cpu.col
```

You may get some error messages, similar to the following when you execute this command:

```
WTR05128:A communications failure occurred: destination dispatcher unavailable
Please refer to the TME 10 Framework Planning and Installation Guide, \
″TME Maintenance and Troubleshooting″ for details on diagnosing \
communication errors or contact your Tivoli support provider.
```

This is not necessarily a serious problem. The command attempts to update all of the managed nodes in the TMR. If there are nodes that are currently down

you will see the above message. However, the monitoring collection will have been installed on all of the active systems.

To see the new collection and its monitors in the graphical user interface, you have to recycle the object dispatchers using the following command:

```
odadmin reexec all
```

### 5.3.4 The New Monitoring Collection in Action

Now we can invoke our new monitors. We open a TME 10 Distributed Monitoring profile, as shown in Figure 72.

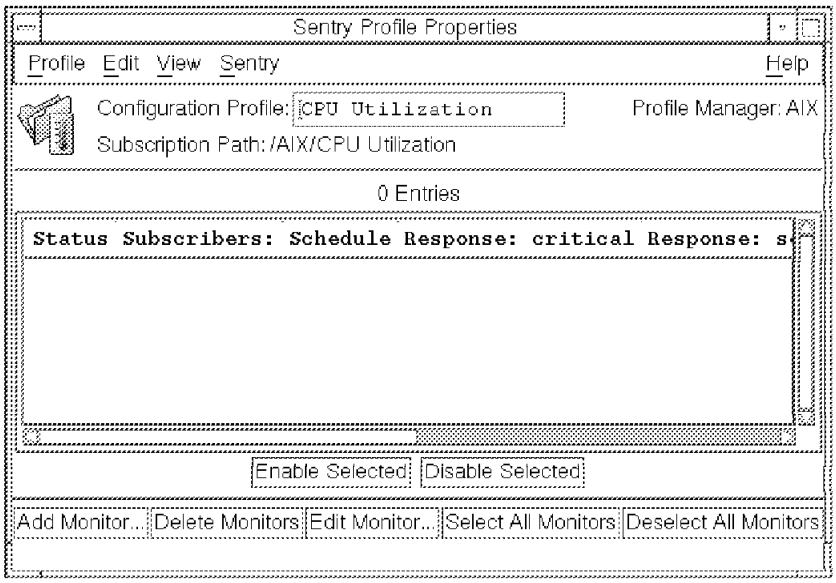



*Figure 72. TME 10 Distributed Monitoring Profile*

Next click on **Add Monitor**. You will be presented with the monitoring collection selection screen (see Figure 73 on page 88). In the left table the new UnixCPU collection will appear. When you select it our two monitors, Current CPU utilization and CPU utilization over time will be listed in the right-hand table.

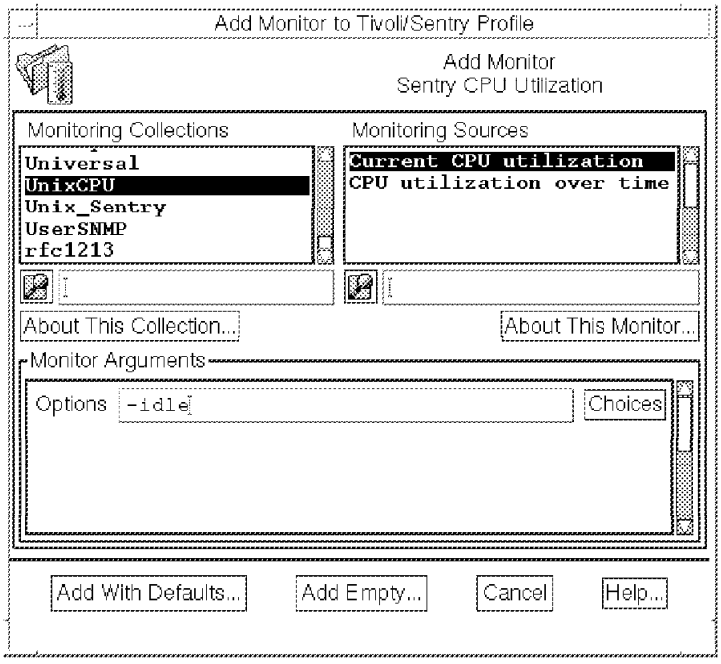



*Figure 73. Adding a New Monitor*

If you now select **Current CPU Utilization** from the Monitoring Sources window and then click on **About This Monitor...** the display shown in Figure 74 will appear. Notice that this is the content of the Busy_Help message catalog entry that we defined in the cpu.msg file (see Figure 69 on page 84). It appears here because we specified:

```
HelpMessage = (cpu_Busy_Help)
```

in the definition of the monitor (Figure 70 on page 85).

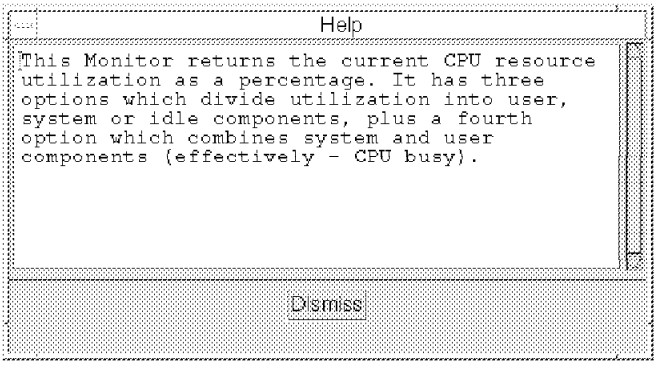



*Figure 74. About This Monitor*

Click on **Dismiss** to close the window and return to the monitoring collection window. Now click on **Choices** beside the Options field to get the list of CPU utilization elements that you can monitor (see Figure 75 on page 89).

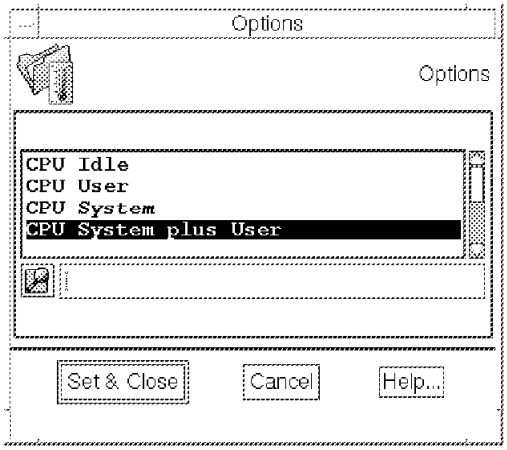



*Figure 75. Options for the Monitor*

Selecting a value from the list and then pressing the **Set & Close** button will close the window and put a value in the text box for the Options field, for example, selecting **CPU System plus User** will put -system_user in the text box.

After adding the monitor by clicking on **Add Empty...**, you are presented with the normal screen for editing the monitor details, as shown in Figure 76.

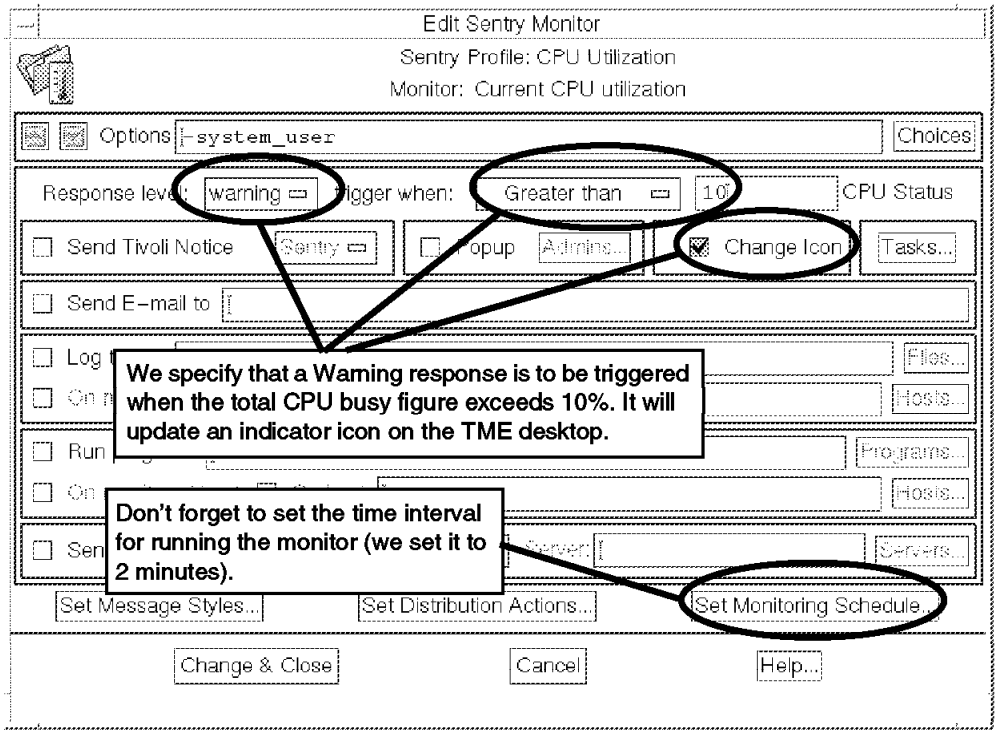



*Figure 76. Trigger Options of the Monitor*

Click on **Change & Close** when you have finished. We specified that the monitor should update an indicator icon, so we have to create an indicator collection and then associate the monitor with it (see Figure 77 on page 90), before saving and distributing the monitor profile.

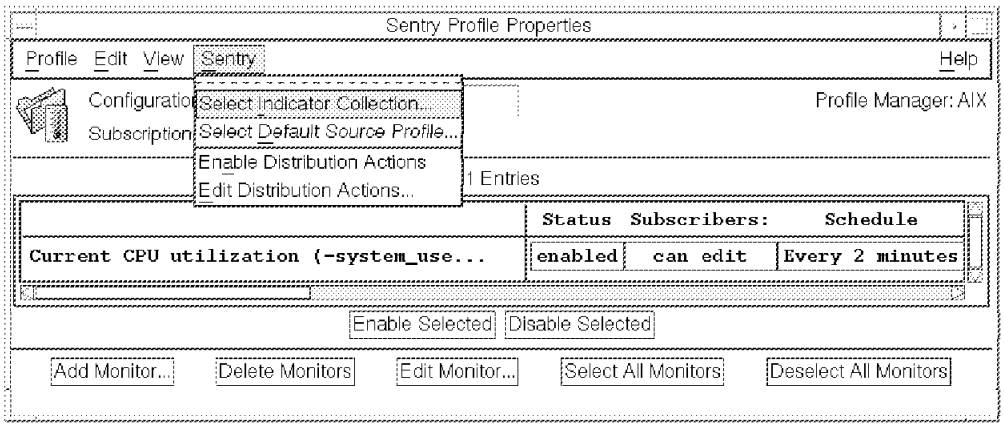



*Figure 77. Assign an Indicator Collection to the Profile*

The result of all of this is that when the combined user and system utilization of one of the monitored nodes exceeds 10%, an icon in the indicator collection will show a warning, as shown in Figure 78.

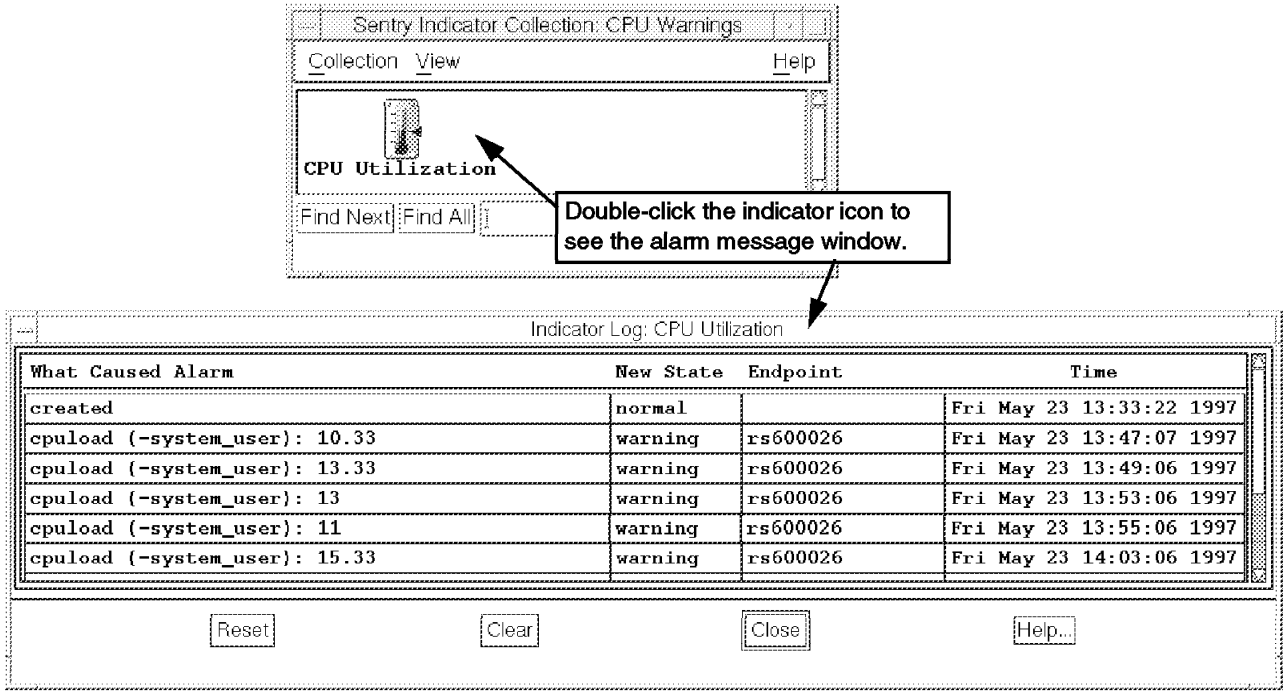



*Figure 78. Operation of the Indicator Icon*

**Remember**

All of the examples in this book are available as a package. See Appendix B, “How to Get the Samples in This Book” on page 141 for details. The CPU utilization monitoring collection described here is merged into a combined collection, called Redbook_samples in the code package.

### 5.3.5 Installing the UnixCPU Collection on Another TMR Server

Once you have successfully compiled and tested the CPU collection, you can install it on another TMR server. We developed the examples on a system running Tivoli/Sentry 3.0.2 and installed the created cpu.col file on a machine running TME 10 Distributed Monitoring 3.5 by invoking the following command on the target system (after transferring the file cpu.col to the target system):

```
mcsl -Ri cpu.col
```

Recycle oserv on the target systems to see the changes.

The reason why we installed our CPU collection on a TME 10 Distributed Monitoring 3.5 system was that we wanted to combine our CPU monitor output with the new *Sentry Graphable Log* TME 10 Distributed Monitoring task. This task saves the results of a monitor in a file and provides a Web-based application to produce a graphical view of the results. For viewing the results, you can use a Web browser, such as Netscape Navigator.

#### 5.3.5.1 Setting Up the Monitors

We created four monitors to collect the CPU utilization data that we want:

*Table 5. Setup for Graphical CPU Monitoring*

| Monitor Type | Monitor Arguments | Monitoring Interval | Actions |
|---|---|---|---|
| Current CPU utilization | -user | 2 minutes | None |
| Current CPU utilization | -system | 2 minutes | None |
| CPU utilization over time | -user 5 | 10 minutes | "always" Response level executes Create Graphable Log task |
| CPU utilization over time | -system 5 | 10 minutes | "always" Response level executes Create graphable log task |

The first two monitors check CPU utilization at 2-minute intervals and write the results to local log files. The third and fourth monitors summarize the collected data over a 10-minute period (five 2-minute intervals) and pass the results on to the graphing facility. Figure 79 on page 92 shows how to assign the graphing task to the monitor.

We then distributed the profile containing the four monitor definitions. The Create graphable log task writes log files to the following directory on the monitored system:

```
$DBDIR/.sntglog/<system name>/UnixCPU/
```

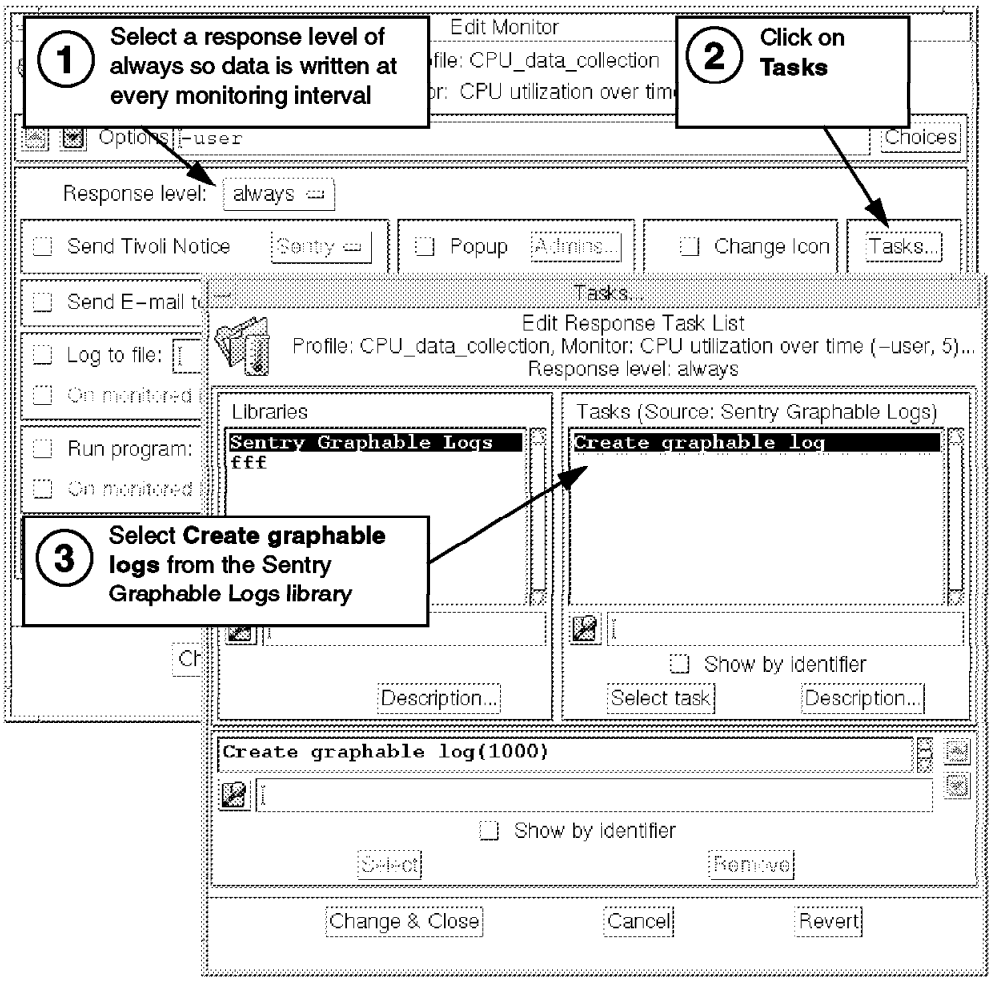



*Figure 79. Selecting the Graphical Monitoring Task*

### 5.3.5.2 Displaying the Logged Data Graphically

The graphical monitoring facility provides a Web-based application for displaying collected results. It includes a specialized Web server, some CGI programs and a set of Java applets. The applets allow you to select the source data and display it on a dynamically updated graph in any Java-enabled browser. If you want to know more about the way that this application works, refer to *A First Look at TME 10 Distributed Monitoring 3.5*, SG24-2112.

For our example we selected to display the system and user components of CPU utilization on one target node as a surface chart. The result is shown in Figure 80 on page 93. Note that this display also shows one of the features of the graphical display; when you stop the mouse pointer over a data point, the value of it pops up in a little box.

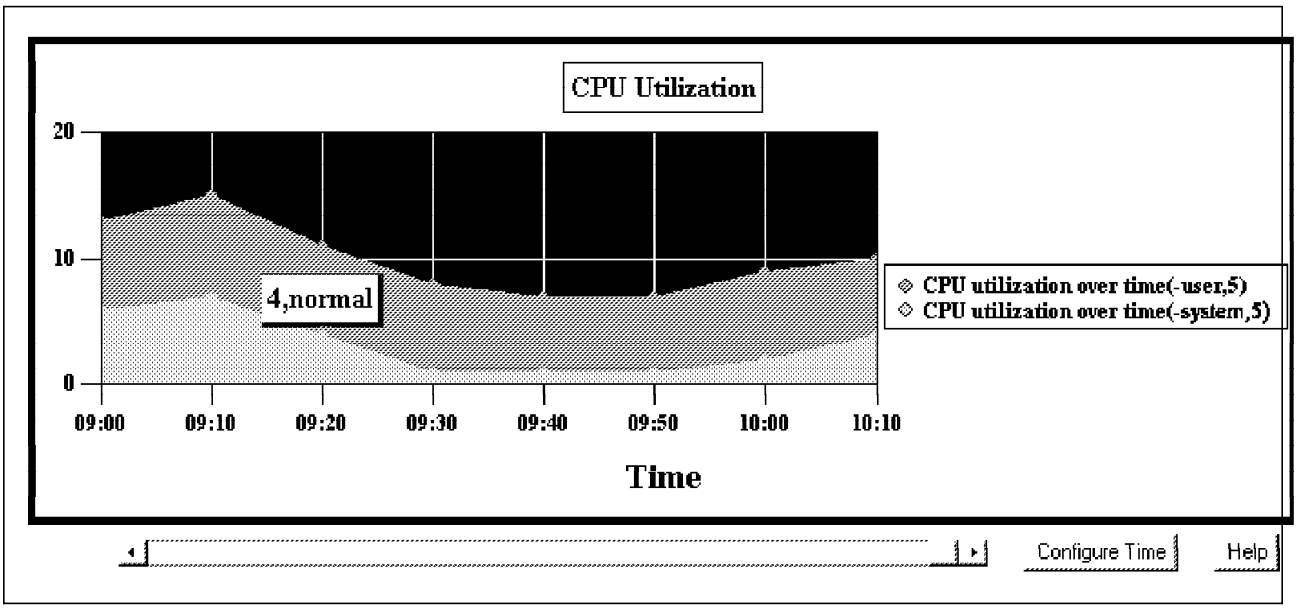



*Figure 80. Graphical View of Historical UNIX CPU Data*

## 5.4 File Monitoring Examples

Under Systems Monitor, file monitoring was rather unusual. For most polling-based functions, the work was performed by the MLM or SLM components, with the SIA as a passive SNMP data source. For file monitoring, however, the SIA performs the polling for status and contents locally. As we showed in 4.3, “Analyzing the Migration” on page 27, the file monitors can be directly migrated, with some caveats. TME also provides us with an alternative method for monitoring file data, using the T/EC log file adapter. We will not consider that option in this book, although in some cases it may be a good alternative to using Sentry monitors.

In this section we will show two examples of file monitoring:

- Monitoring ADSM log files
- Dynamically named log files

### 5.4.1 Monitoring ADSM Log Files

Many customers use ADSTAR Distributed Storage Manager (ADSM) on various platforms to fit their backup and restore needs. Tivoli offers a TME 10/Plus Module for controlling and monitoring ADSM using TME. In this example, however, we will use TME 10 Distributed Monitoring to directly monitor the ADSM log files.

On AIX machines, there are three log files that we are interested in:

- dsmsched.log (sched.log)
- dsmerror.log
- the console error log

Looking at the output of the log files, they all have a similar structure.

The following is an example of the dsmerror.log file:

```
11/06/1996 13:50:18  sessOpen: Error 137 from signon authentication.
11/06/1996 13:50:18  ANS4028E Session rejected: Authentication failure
```

Typical entries from the dsmsched.log look like the following:

```
11/06/1996 07:50:16 Querying server for next scheduled event.
11/06/1996 07:50:16 Next operation scheduled:
11/06/1996 07:50:16 ------------------------------------------------
11/06/1996 07:50:16 Schedule Name:         DAILYINCRNT
11/06/1996 07:50:16 Action:                Incremental
11/06/1996 07:50:16 Objects:
11/06/1996 07:50:16 Options:
11/06/1996 07:50:16 Server Window Start:   19:00:00  on 11/06/1996
11/06/1996 07:50:16 ------------------------------------------------
11/06/1996 07:50:16 Schedule will be refreshed in 6 hours.
11/06/1996 13:50:18 ANS4028E Session rejected: Authentication failure
11/06/1996 13:50:18 Scheduler has been stopped.
```

Finally, the console error log file contains entries like the following:

```
11/06/1996 13:55:40  ANR0400I Session 2270 started for node SORL0008 (WinNT ) ref
11/06/1996 13:55:40  ANR0424W Session 2270 for node SORL0008 (WinNT) ref \
invalid password submitted.
11/06/1996 13:55:40  ANR0403I Session 2270 ended for node SORL0008 (WinNT) ref
```

ADSM operates on many different operating systems. The messages that it places in the log files are generally the same, regardless of the platform. However, there are some system-specific messages; for example, the following log file excerpt is from ADSM for MVS, showing system messages interpolated with related ADSM messages:

```
ANR0352I Transaction recovery complete.
ANR2100I Activity log process has started.
IEF237I 0728 ALLOCATED TO SYS00009
ANR2102I Activity log pruning started: removing entries prior to 02/23/1997
00:00:00.
IEF237I 0728 ALLOCATED TO SYS00010
ANR2103I Activity log pruning completed: 8 records removed.
IEF237I 0728 ALLOCATED TO SYS00011
ANR1305I Disk volume ADSM.STORAGE.POOL001 varied online.
ANR5030E Dynamic allocation of data set ADSM.STORAGE.POOL007 failed, \
return code 4, error code 1156, info code 0.
ANR1305I Disk volume ADSM.STORAGE.POOL003 varied online.
ANR1311E Vary-on failed for disk volume ADSM.STORAGE.POOL007 - unable \
to accessdisk device.
IEF237I 0728 ALLOCATED TO SYS00016
ANR1305I Disk volume ADSM.STORAGE.POOL004 varied online.
ANR0811I Inventory client file expiration started as process 1.
ANR2803I License manager started.
```

By looking at these examples, it should be clear that there is always at least one line (sometimes two or more) with an error code (for example, ANR5030E, ANR1311E) and a brief description message.

Messages in ADSM always start with ANR, followed by a message number and the message class. The message number has four digits, and the message class can be, for example, ″I″ for information messages or ″E″ for errors.

### 5.4.1.1 Monitoring with Systems Monitor

It is quite easy to tell Systems Monitor to look at the output file for all errors. We can create a file monitor and look for the following regular expression to find all error messages in the log file:

```
ANR[0-9][0-9][0-9][0-9]E
```

If a relevant error message is found, we can use the snmptrap command to send an SNMP trap to NetView passing the complete error message line. This can be achieved by using the $SM6K_FILE_MONITOR_COMPLETE_LINE_STRING_FOUND variable in Systems Monitor when calling the snmptrap command.

However, since there are some errors that occur very often and are not interesting at all, we decided to add some more functionality to the basic monitor. We created an exception list that contains only the error messages that are relevant to us. The monitor can then use this exception list and only send a trap if the error that was found in the message log is in the exception list. Figure 81 shows the monitoring script.

```
#!/bin/ksh
#
# If we get here, Filemonitoring has found a 'ANR????E'-string in the
# logfile.
# go through the exception list, whether this string should be sent or not.

complete=$*
exceptions='./exceptions'
netviewmachine='rs600026.itso.ral.ibm.com'

for exception in `cat $exceptions`
do
  result=`echo $complete |grep $exception`
  [ "$result" ]
  if [ $? -eq 0 ]
  then
    # no need to go on further
    exit 0;
  fi
done
/usr/lpp/smsia/original/snmptrap $netviewmachine public \
".1.3.6.1.4.1.2.6.12" `hostname` 6 41 0 ".1.3.6.1.4.1.2.6.12" \
OctetStringASCII "$complete";
/usr/local/Tivoli/bin/aix4-r1/bin/wsndnotif "TME Diagnostics" Error <<EOF
ADSM-script found Error Entry on host `hostname`
$complete
EOF
exit 1;
```

*Figure 81. adsm_script Monitoring Script for Systems Monitor*

The exception list is a plain text file and contains entries like the following:

```
ANR1234E
ANR8888E
ANR5435E
```

The following figure shows how we set up the monitoring for the dsmcon.log file in Systems Monitor.

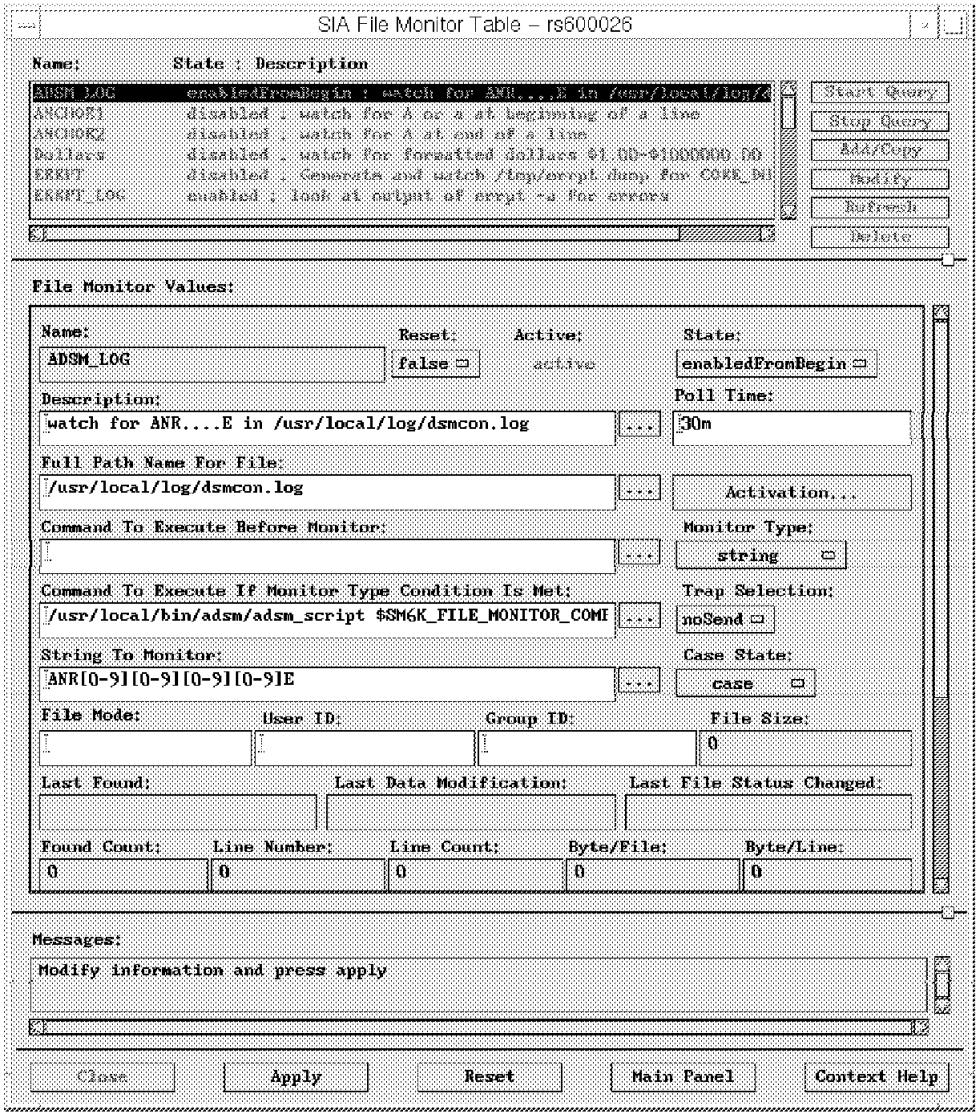



*Figure 82. Systems Monitor Configuration for ADSM File Monitoring*

In our example we defined an enterprise-specific trap ID 41 for our monitor to create. The event that is displayed in the TME 10 NetView Control Desk window when the monitor triggers is shown in Figure 83 on page 97.

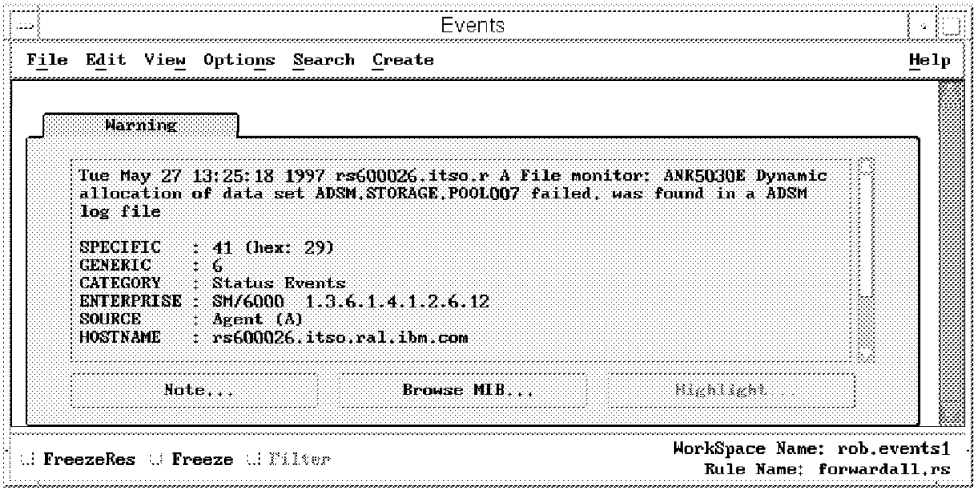



*Figure 83. SNMP Trap Generated by ADSM File Monitoring*

### 5.4.1.2 Migrating the ADSM Monitor to Sentry

We chose to use the migrate_sysmon_config shell script from the sample package for this book to convert the SIA file monitor into a Sentry profile (see “Semi-Automated Profile Migration” on page 46 for a discussion about this). To do this, we copied the SIA configuration file from its default location in directory /var/adm/smv2/sia/config into a working directory and then invoked the command. The command dialog follows:

```
rs600026:/tmp/migtest > migrate_sysmon_config
You are Tivoli administrator Root_ssd-region
total 136
drwxr-xr-x   2 rob      assignee    2048 Jun 05 10:51 .
drwxrwxrwt  12 root     system      3584 Jun 05 10:52 ..
-rwx------   1 root     system       862 Jun 05 09:31 itso.config.test
-rw-r--r--   1 root     system     19384 Jun 01 07:09 kuffi
-rwxr-xr-x   1 root     system     21434 Jun 01 08:20 m_s_c
-rw-r--r--   1 rob      assignee    4449 Jun 05 08:30 mlm_plus_sia
-rw-r--r--   1 root     system       586 Jun 05 10:51 sia_filemon


Please input the Sysmon config name from the list above ==>sia_filemon
Please input Sentry profile name in which to place migrated monitors
Note: this profile must already exist ==>adsmtest
Path to monitoring scripts on monitored system [/usr/local/bin] ==>
/tmp/migtest/sia_filemon
Creating countstr monitor for 'ANR[0-9][0-9][0-9][0-9]E' /usr/local/log/dsmcon.log

Migration Completed - see /tmp/migtest/migrate_sysmon.log for details
rs600026:/tmp/migtest > cat migrate_sysmon.log
################## Thu Jun  5 10:53:03 EDT 1997 ############################
SYSMON SCHEDULE====30m
SENTRY SCHEDULE====30
Creating countstr monitor for 'ANR[0-9][0-9][0-9][0-9]E' /usr/local/log/dsmcon.log
============================================================
```

The TME 10 Distributed Monitoring definition created by this command is shown in Figure 84 on page 98. Note that the equivalent test to the SIA string monitor is an *Occurrences in file* monitor, with a criterion of ″Changes by 1″. There are a number of other points to note here:

- The new monitor is created in a disabled state. You should check that it has been migrated successfully before enabling it.

- The migration script, by default, defines a pop-up message as the monitor action. You will probably want to change this, depending on your local policy.
- The command to execute from SIA has been migrated as a *Run program* action. This will often be acceptable, but this case is an example where the program uses Systems Monitor environment variables. The command line would therefore need to be modified to work correctly with Sentry. In this case the variable contains the line from the log file that caused the monitor to be triggered. Unfortunately there is no exactly equivalent Sentry environment variable.

The lesson from this is that the sample migration scripts are a useful way to create a set of Sentry monitors that reflect your Systems Monitor environment. However, the monitors it creates are only a foundation, which you will have to build upon to create the final solution.

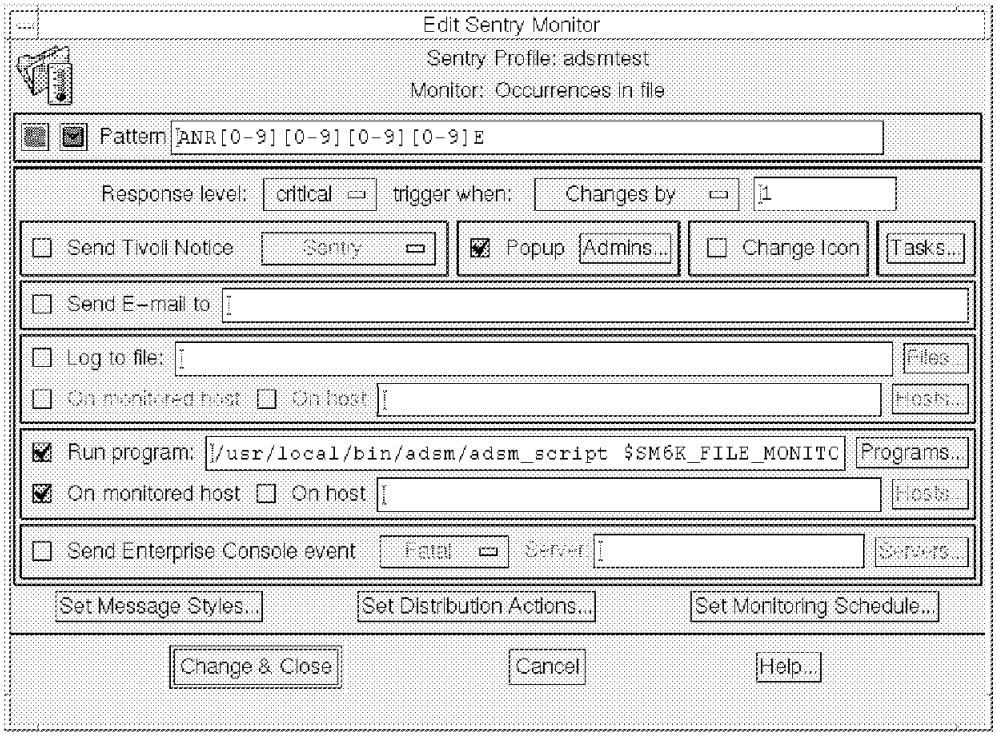



*Figure 84. Migrated File Monitor*

The pop-up generated by the migrated monitor is shown in Figure 85 on page 99.

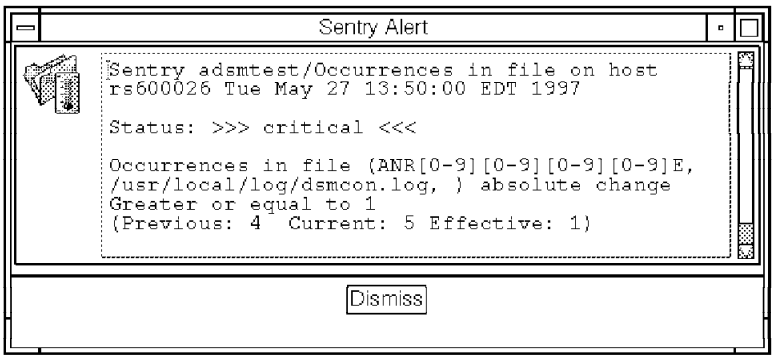



*Figure 85. Pop-up Notification of ADSM Error Message*

## 5.4.2 Dynamically Named Log Files

Some applications create log files with the date added at the end of the log file name, for example, by appending the output of the AIX date command to the end of the filename. One example of this is SysBack/6000. We show how monitoring of these log files can be handled.

SysBack/6000 is a backup and restore application that can be configured to create a log file containing details of all the files that were backed up and the errors encountered, if any. For example, a customer runs a daily backup of critical file systems and on Saturday does a full backup of the entire volume group. The log files are written to /usr/local/log/sysback.log.‘date +%d%m%y‘.

After a few days there would be a number of files, with names like sysback.log.010397, sysback.log.020397, sysback.log.030397 and so on.

Every day, after the backup has completed we want to monitor the actual log file to determine if anything unusual has happened.

### 5.4.2.1 Monitoring with Systems Monitor

The problem is that the Systems Monitor file monitor table needs a fixed file name to look for. In order to face this problem we can look for the newest file that matches the naming conventions for the log file. If this file is newer than the file with the fixed name (sysback.log in our case) that we want to monitor, it is copied to sysback.log and Systems Monitor will then look at this file.

The following code fragment can be used to achieve this:

```
dest='/usr/local/log/sysback.log'
file=`ls -tr /usr/local/log/sysback.log.* | tail -1`
if [ $file -nt $dest ]; then cp $file $dest; fi
```

What we want, therefore, is a way to be sure that the above script has been executed before testing the log file contents. SIA provides us with the *command to execute before monitor* option which does exactly what we want. Figure 86 on page 100 shows an example of this, in which we are looking for the words ERROR or WARNING in the latest log file. Notice that we have specified that the monitor is only activated once per day, so that we do not process any log events a second time.

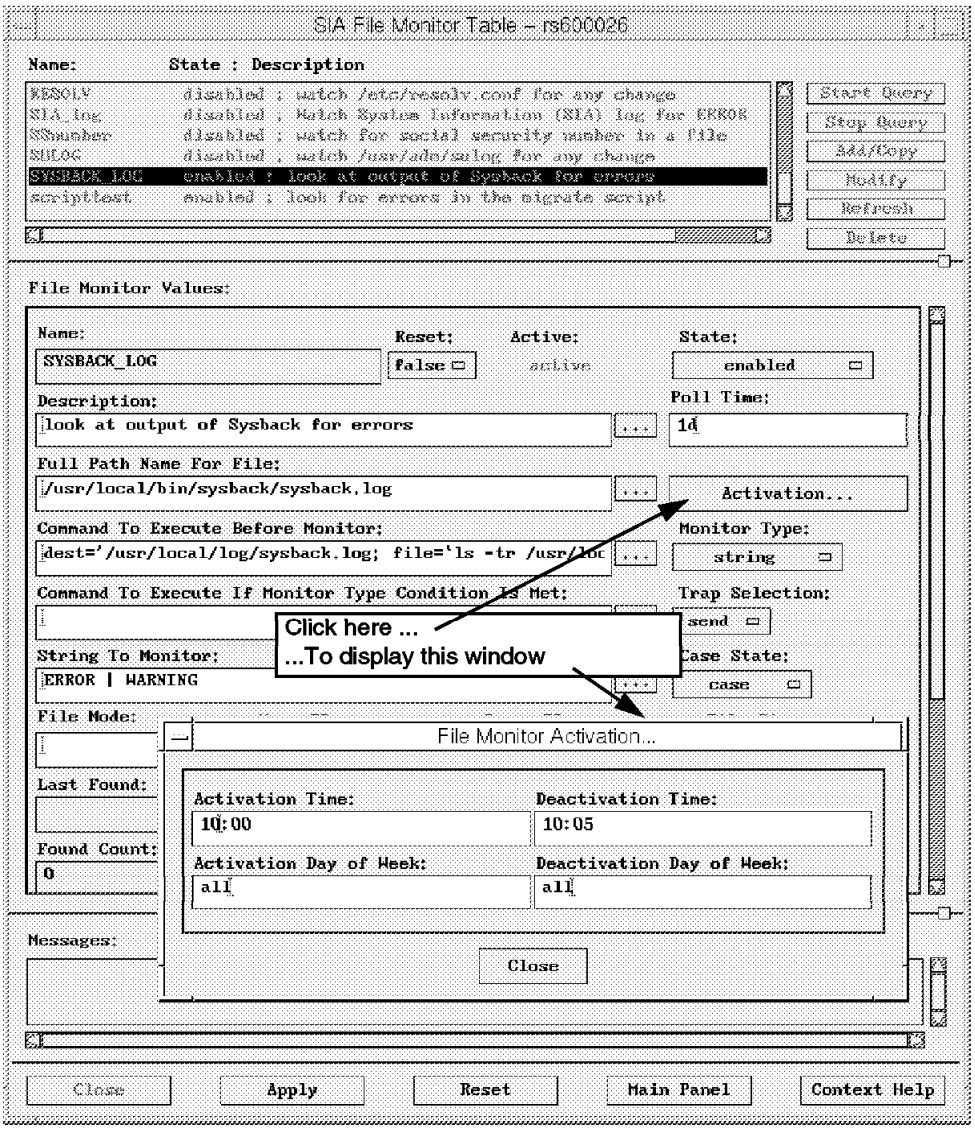



*Figure 86. Monitoring Dynamically Named Log Files with Systems Monitor*

### 5.4.2.2 Monitoring with and/or Migration to Sentry

TME 10 Distributed Monitoring (Sentry) does not have a standard option that allows a command to be run before the monitor. This is why we cannot employ an automatic migration of the above example.

A solution to the problem is to use two monitors:

- A *numeric script* monitor from the universal collection, which does the sorting and copying to a file with fixed filename
- An *occurrences in file* monitor from the Unix_Sentry collection, which checks for the file pattern matches

The script for the first monitor is exactly like the three lines of code shown in the previous section. The second monitor would normally be a simple one, but in this case the pattern that we want to watch for ( ERROR | WARNING) is an *extended regular expression*. The Sentry monitors use grep-style limited regular expressions, so the SIA monitor cannot be directly migrated. To resolve this

problem, we created a new monitoring collection, containing a file monitor that uses egrep for pattern matching. The collection is available in the code package for this redbook. Figure 87 on page 101 and Figure 88 on page 101 show the two monitor definitions. You can see that we have set them to trigger at a specific time each day, one 5 minutes after the other.

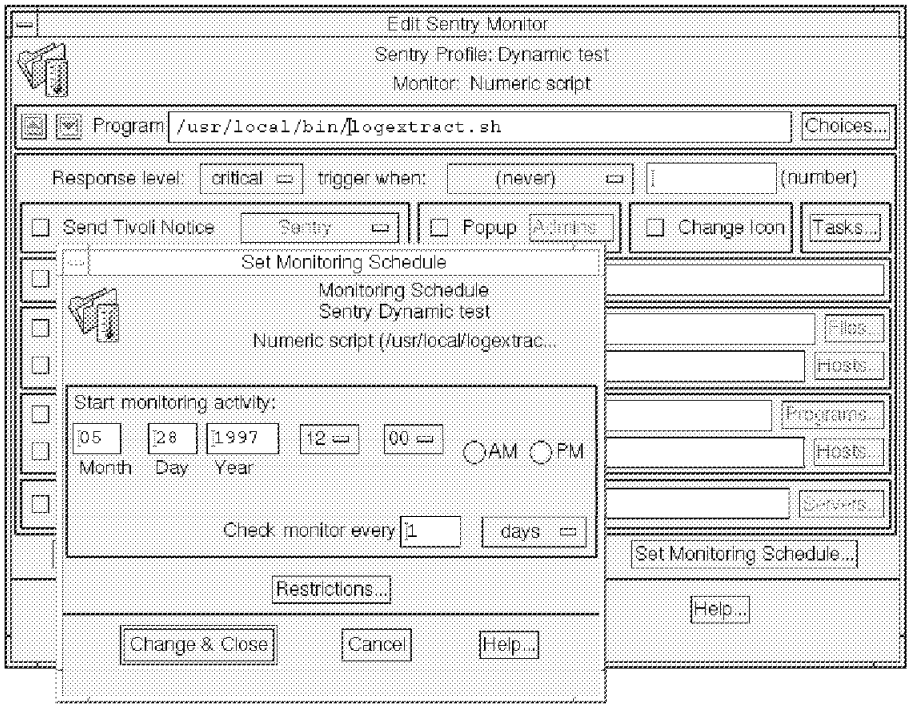



*Figure 87. First Monitor Extracts Daily Log File*

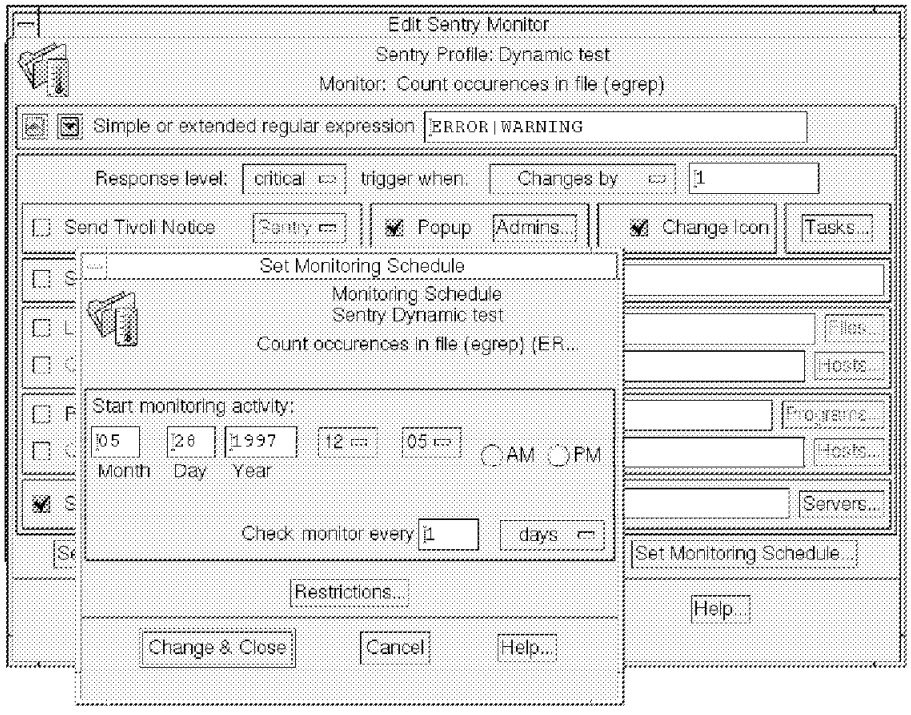



*Figure 88. Second Monitor Analyzes Log File for Error Messages*

## 5.5 Migrating the Re-Arm Function

One of the features of Systems Monitor that TME 10 Distributed Monitoring does not provide is the use of threshold and re-arm values to control automated actions and alerts. Figure 89 illustrates this principle.

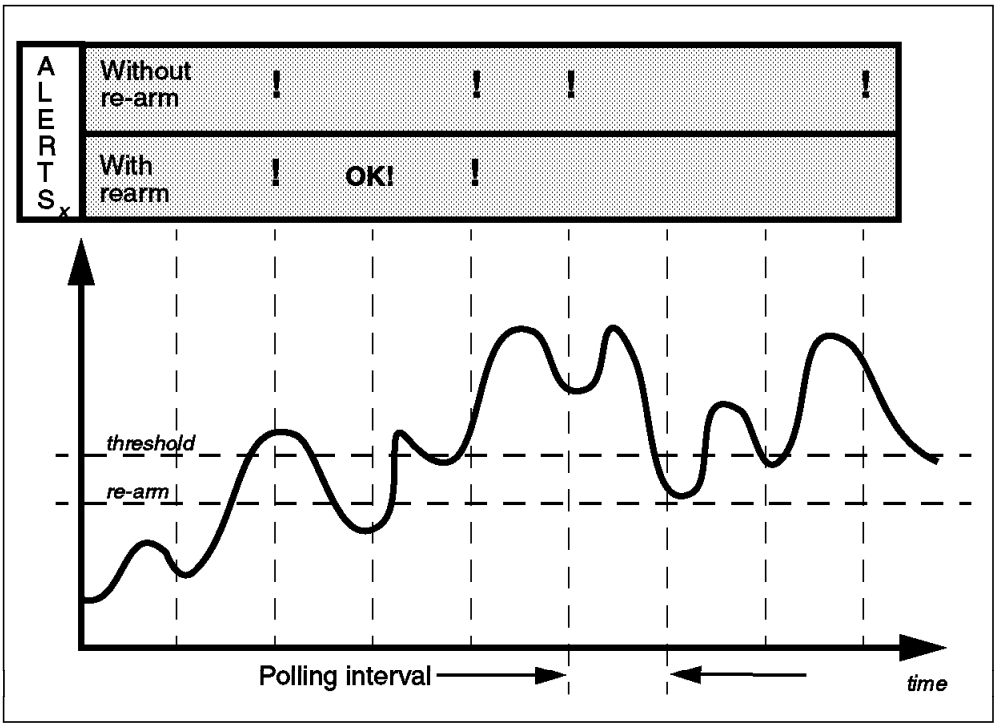



*Figure 89. Threshold and Re-Arm Function*

With a simple threshold mechanism, the monitor will be triggered at every polling interval for which the value exceeds the threshold. With a re-arm mechanism you can set a re-arm line, below the threshold level. Once the threshold value is exceeded by the value of x, an event is sent. In following probes no further events are sent even if the threshold is still exceeded until the value of x drops below the re-arm line. Then the monitor is ″re-armed″ and the next time x exceeds the threshold another alert is sent. This has the dual benefit of reducing the number of alerts and also providing a resolution event, to indicate that the problem episode has come to an end.

If the threshold and re-arm levels are the same, one event is sent when x exceeds the threshold and then monitoring is disabled. Once x drops below the threshold again, the monitor is ″re-armed″. The re-arm capability can also be applied to non-numerical monitors. For example, if you are monitoring the status of daemon processes, you can set a threshold to generate an alert when a process fails and set it to re-arm when the process has restarted.

## 5.5.1 An Approximate Equivalent of Re-Arm Using Sentry

As we have shown, the re-arm capability can be a very useful feature of Systems Monitor. How should we provide an equivalent function in TME 10 Distributed Monitoring? The obvious answer is that the ″normal″ response is similar to the resolved response that re-arm gives you. The problem with this is that it is *always* triggered whenever any of the exception responses (critical, severe, warning) are not triggered. This means that you will be sending monitor events at every polling interval, which is a good way to use a lot of system resource and bandwidth.

The best solution is to exploit the flexibility of Sentry monitors by, for example:

1. Using the additional relative test conditions that they offer (such as *increases beyond*, *decreases below*, *becomes active etc.)*
2. Using their capability to add response levels with the waddlevel command

As an example of this, we will create a monitor that generates an alarm when the utilization of a given file system exceeds 90% and then generates a resolution event when it falls below 80%.

### 5.5.1.1 Adding the Resolved Response Level

Sentry monitors normally have three exception levels, critical, severe and warning. Any monitor that does not trigger any of these is considered to be at the *normal* response level. We want to add a further level, *resolved*, which will be logically below the warning level but above the normal level.

The waddlevel command allows us to do this. To add our resolved level to a monitoring profile called rearm_test, we entered the following command:

```
waddlevel resolved 10 rearm_test
```

The numerical value, 10, in this command is the *precedence* of the new level. This is an arbitrary value that falls between warning, with a precedence of 100, and normal (0). What this means is that if both the warning and the resolved thresholds are triggered, it is the warning action that will be invoked.

### 5.5.1.2 Defining the Monitor

Now we can use the new response level to implement our file system monitor. Figure 90 on page 104 shows how to do this. Notice that we have used the increases beyond and decreases below comparators to trigger the two response levels. Both response levels are defined to update a monitoring collection icon, but only the Warning event will actually change the icon. This is because the icon needs to have all of its possible states predefined, with a bitmap and dialog associated with each state. The wputicon command can be used to add a new state, but we did not try it for this project.

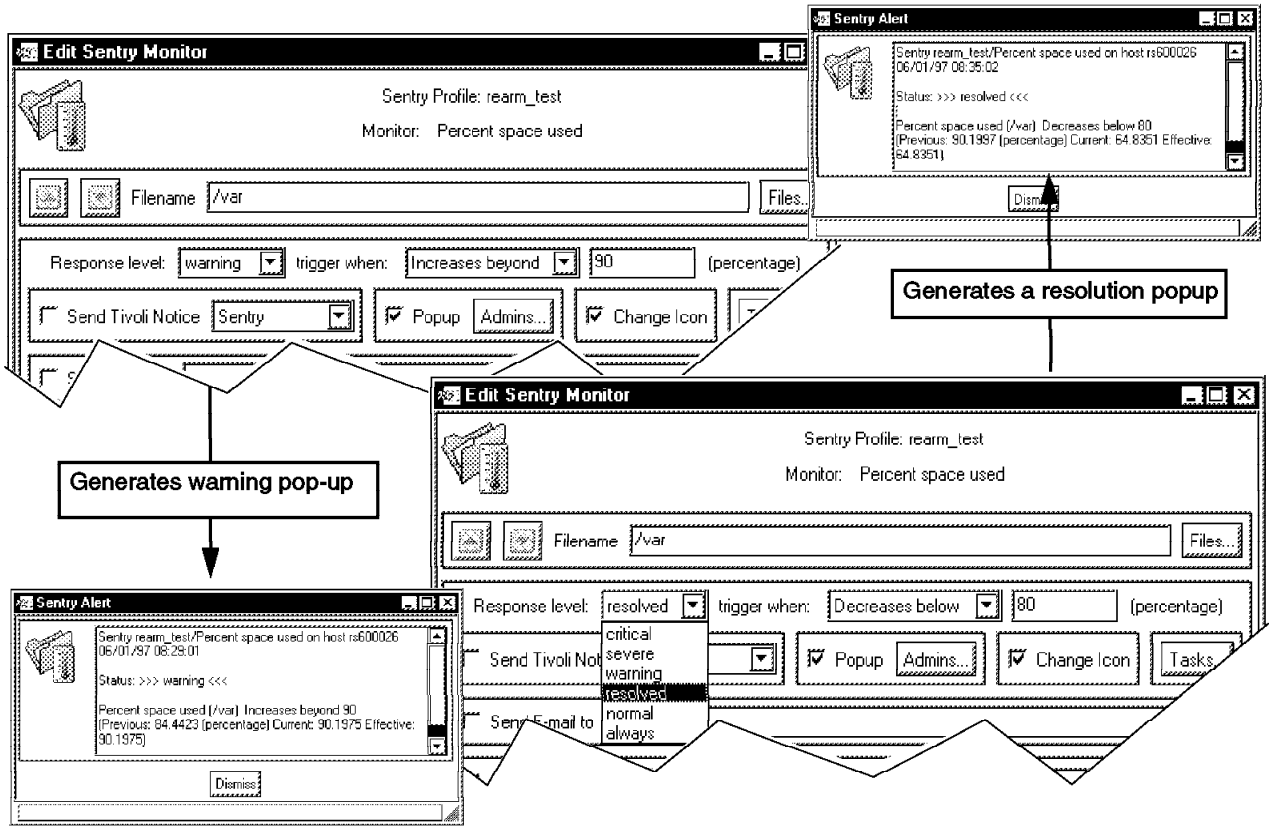



*Figure 90. Creating a Re-Arm-Like Capability with Response Levels*

## 5.5.2 A Closer Approximation to Re-Arm Using Sentry

Adding a resolved response level, as described above, gives very similar results to the Systems Monitor re-arm function. It differs in one way: if the monitor value exceeds the alarm level and then drops, but *not below the resolved level*, it will generate another alarm event when it next exceeds the level. By contrast, re-arm causes a new alarm to be triggered only when the monitor has first dropped below the resolved level.

In most cases, this slightly different behavior is not a problem and we recommend using this approach. However, if you want to replicate the re-arm function more precisely, the script shown in Figure 91 on page 105 will allow you to do so.

```
#!/bin/ksh
#########################################################################
# FILE:         sentry_rearm
#
# This script is an example from IBM redbook SG24-4936. It is made freely
# available on the understanding that it is unsupported sample code only.
#
# DESCRIPTION:
# This script replicates the re-arm capability of Sysmon threshold table and
# file monitor table entries in Sentry.
# The script should be run as an action from a sentry monitor. It alters the
# response level of critical, severe or warning temporarily, until a "resolved"
# response resets it to the normal value.
#
# ARGUMENTS:
# When invoked from critical, severe or warning, 2 arguments required.
# When invoked from resolved, 3 arguments required
# (1) The new comparator for the response level (ie: "->>" for increases beyond)
# (2) The new threshold level to be compared against
# (3) The response level to set (only needed for "resolved" action)
#
# For example:
#
# On the critical response level, execute:
#    /usr/local/bin/sentry_rearm "->>" 80   to set the new "increases beyond"
#                                           value to 80
# On the resolved response level, execute:
#    /usr/local/bin/sentry_rearm "->>" 90 critical to reset the threshold to 90
#
# RESTRICTIONS:
# This script uses the "execute program" option for the monitor response levels,
# so you cannot have another program executed.
# It also needs appropriate TME administrator authority to run.
#
# AUTHORS:      Graeme Naysmith IBM UK
#               Rob Macgregor ITSO-Raleigh
#########################################################################

# Note: uses environment vars supplied by Sentry: $NAME, $HOSTNAME and
# $MONITOR and $RESPONSELEVEL. Would use $INTERNAL_ID, but could not make
# it work in this version of Sentry, so extract it using wlsmon instead.

# Get the internal monitor ID
wlsmon $NAME"@"$HOSTNAME  | grep "$MONITOR($PROBE_ARG" | awk '{print $1 }' | read INTERNAL_I
D

# Check if a rearm or an arm
if [[ $RESPONSE_LEVEL != "resolved" ]]
then
  wsetmon -c $RESPONSE_LEVEL -R "$1" "$2" $INTERNAL_ID $NAME"@"$HOSTNAME >&1
else
  wsetmon -c $3 -R "$1" "$2" $INTERNAL_ID $NAME"@"$HOSTNAME
fi

wdistrib -l over_all @SentryProfile:$NAME"@"$HOSTNAME @ManagedNode:$HOSTNAME

exit
```

*Figure 91. An Example of Providing Re-Arm within a Sentry Monitor*

The first thing the script does is to determine the internal ID of the monitor and store it in the $INTERNAL_ID variable since this is needed with the wsetmon command. The wsetmon command is then used to update the monitor, but note that it only updates the copy on the monitored system. This means that the threshold value will be reset only on the system that has the problem, not on the base Sentry profile. Finally the script updates the sentry_engine on the monitored system using the wdistrib command.

To use the script, you must run it as an action from the monitor response level. Figure 92 on page 106 shows how to invoke sentry_rearm for the same test that

we used before (a file system percentage full monitor for /var). In addition to adding the command invocation to the response levels, you must also check that the monitor is running with a user ID that has the necessary access to the monitor profiles. See 5.2.2, “Automating Daemon Recovery” on page 67 for a description of how to do this.

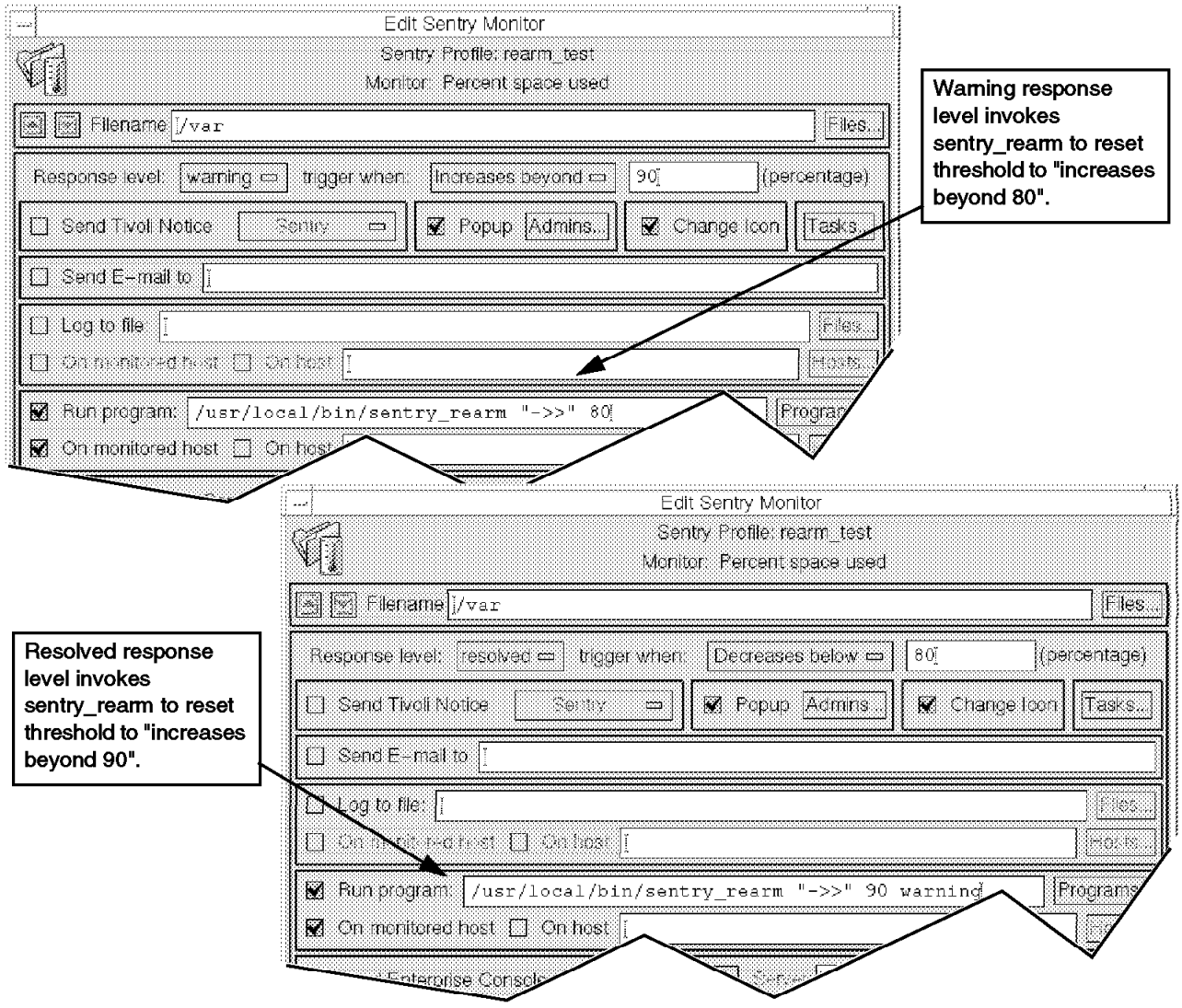



*Figure 92. Defining Monitors to Use the sentry_rearm Script*

The monitor behavior that results from this definition is as follows:

1. The utilization goes above 90% for the first time. This causes an alert to pop up and invokes sentry_rearm with arguments ″- > > ″ 80 (*increases beyond* 80).
2. The utilization drops below 90%, but not below the re-arm level of 80%.
3. The utilization increases again above 90%. Nothing happens, because the threshold condition (increases beyond 80) has not been met.
4. The utilization falls below 80%. The resolved response condition is *decreases below* 80, so a resolved message pops up and sentry_rearm is invoked with arguments ″- > > ″ 90 warning. This sets the warning response level back to increases beyond 90 again, thereby restarting the cycle.

## 5.6 Advanced Process Monitoring

In this section we show some examples for advanced process monitoring, namely:

- Monitoring multiple processes
- Monitoring process groups

The first example is a script that can be used with both Systems Monitor and TME 10 Distributed Monitoring, the second example will be implemented in a monitoring collection using TME 10 Distributed Monitoring's MCSL language.

### 5.6.1 Monitoring Multiple Processes

The applications running on a UNIX server can be very complex, involving a number of related processes. Often we would like to monitor a number of processes as a group, as opposed to monitoring single processes.

We created a shell script that will look for a list of processes being up or down. The first time a process is found to be down, an event is sent to a management platform (this is TME 10 NetView in our example but could be T/EC also). We do not alert multiple times but send a message if the process is up again.

The script is shown in the following figure:

```
#!/bin/ksh
export TRAP=/usr/local/bin/sendtrap
host_name=`hostname`
pscheckdata=/usr/local/log/pscheck.dat
logstart=/usr/local/log/process_check.log
# Check that the processes listed in pscheck.dat are running.
# make a list of all processes
pstext=$(ps -e | awk '{ print $4}' | sort | uniq )
# check all lines in pscheck.dat exlcuding lines starting with a hash
cat $pscheckdata | while read i
do
echo $i | grep -v "¬#" > /dev/null 2>$1
if [[ $? -eq 0 ]]
then

   echo $i | cut -d: -f1 | read process
   echo $i | cut -d: -f2 | read alert
   echo $i | cut -d: -f3 | read alert_text

   echo $pstext | grep $process > /dev/null 2>&1
   if [[ $? -eq 0 ]]
      then
      if test -f "/usr/local/log/pscheck.$process"
then
         print `date` "$process is now running" >> $logstart
 $TRAP 7 "$process is now running"
         rm -f /usr/local/log/pscheck.$process
      fi
   else
      # make a note that this process is dead so we remember not to
      # alert multiple times
      if test -f "/usr/local/log/pscheck.$process"
         then
         print `date` "$process is still not running, no alert sent" >> $logstart
         continue
      else
        print `date` "$process is not running - alert sent" >> $logstart
        $TRAP $alert "$alert_text"
         touch /usr/local/log/pscheck.$process
      fi
   fi
fi
done
```

*Figure 93. Shell Script to Check Multiple Processes*

In order to keep the script flexible the processes that are to be monitored are specified in a configuration file called /usr/local/log/pscheck.dat. An example of such a configuration file is shown below.

```
# Check for inactive processes
# Format is
# process name:Alert Number:Alert Text
#
# Process checking for
#
syslogd:2:syslogd is down
sendmail:2:sendmail is down
```

*Figure 94. Configuration File for check_processes*

The script reads the configuration file and then checks for the existence of each of the processes specified in the configuration file. The first time the script finds that the process is either running or not, it sends an SNMP trap to TME 10 NetView. For this the script assumes that the appropriate form of the snmptrap

command is stored in the /usr/local/bin/sendtrap shell script. This script will send the trap to the appropriate NetView machine, community, etc.

Once a trap has been sent, a lock file is created. The next time the same state is found again and the lock file exists, no further trap is sent until the state changes again and the lock file is erased.

### 5.6.2 Monitoring Process Groups

A system administrator often has requirements for process monitoring that are not covered by standard monitors. We show you a new monitor that is able to do some advanced process monitoring and two examples of how parameters can be set.

Think of an Oracle RDBM system. Usually, there must be at least five database processes running, ora_reco_, ora_smon_, ora_lgwr_ and ora_dbwr_. For this example it is sufficient to know that these processes must be running and if at least one is missing, a note to the database administrator will be sent.

The following command will check the number of Oracle processes:

```
ps -ef | grep -y "ora_[reco|dbwr|lgwr|smon|pmon]_" | wc -l
```

If the result of the above command is 5, everything seems to be OK. Since the standard Universal monitor for Application Status delivered with TME 10 Distributed Monitoring does not deal with wildcards, pipes or other special characters we have to write a new monitor with such capabilities.

The second example deals with looking for processes with flags. Sometimes it is not enough to know that a process with a given name is running. There may be multiple copies of that executable running, with different flags or command line arguments. ADSM is an example of this. You want to make sure that your scheduled ADSM process is up and running. The process line we are looking for is:

```
dsmc sched -pass=
```

Once again the standard monitors provided by Systems Monitor and Sentry do not handle this.

Our idea is therefore to create a monitor with two parameters. The monitor will execute a command of the form:

```
ps $Argument1 | grep $Argument2 | grep -v grep | wc -l
```

The variable Argument1 holds the parameters passed to the ps command itself, while the variable Argument2 contains the search string for the ps output.

In the example where we want to monitor Oracle, Argument1 is -ef and Argument2 is:

```
-y "ora_[reco|dbwr|lgwr|smon|pmon]_"
```

Look at Figure 95 on page 110 to see how this is specified in the monitor we will create in this example.

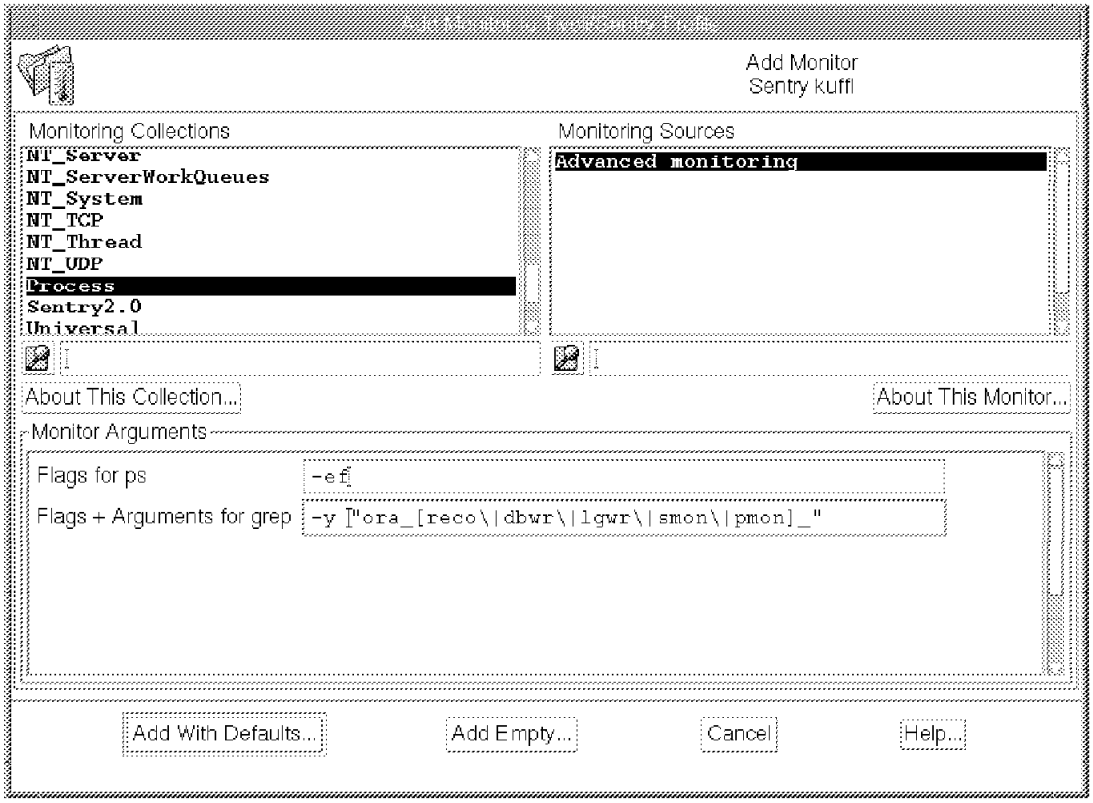



*Figure 95. Example of Monitoring Process Groups, Database Processes*

In the example, monitoring ADSM, Argument1 is again -ef. Argument2 is:

```
"dsmc sched -pass=""
```

We will create the new monitor using MCSL as we did before for the CPU monitoring example (see 5.3.3, "Monitoring UNIX CPU Utilization Using MCSL" on page 81). We will not explain the MCSL syntax in detail here.

```
$ key=generic_help
1 This collection deals with extended process monitoring
$ key=Descr_imbedded
2 Advanced monitoring
$ key=val_Descr
3 Process Status
$ key=Help
4 This Monitor checks the availability of a process, optionally \
with additional flags. It returns the number of running processes which fit the arguments
$ key=ps_flags
5 Flags for ps
$ key=grep_list
6 Flags + Arguments for grep
```

The message file for the new monitor is shown below:

*Figure 96. process.msg File*

To compile the message file, type:

```
gencmsg process.msg
```

```
 +1
 +2  #include "Sentry2_0.dsl"
 +3  #include "process.dsl"
 +4
 +5  Collection "Process" {
 +6      CodeID = "$Id:process.csl,v 1.0$";
 +7      Version = "1.0";
 +8      Require = ">2.0.2";
 +9      HelpMessage = (process_generic_help);
+10      EventBaseClass = "Sample_Sentry_Monitors";
+11      NoticeGroup = "Sentry";
+12      NoticeGroup = "Sentry-log";
+13      NoticeGroup = "Sentry-urgent";
+14
+15  #include "operators.csl"
+16  #include "choicelists.csl"
+17  #include "formats.csl"
+18
+19      Monitor imbedded Numeric Group numeric {
+20        Description = (process_Descr_imbedded);
+21
+22        ValueDescription = (process_val_Descr);
+23        HelpMessage = (process_Help);
+24        Argument (process_ps_flags)
+25        DefaultValue "-ef";
+26        Argument (process_grep_list);
+27
+28        Implementation (aix3-r2, aix4-r1)
+29        Shell("/bin/sh", "-c", Command, "process")
+30        .a=$1
+31        .b=$2
+32        .echo ps $a \| grep $b \| grep -v grep \| wc -l > /usr/local/log/grep.in
+33        . (. /usr/local/log/grep.in)
+34        ;
+35     };
+36  }
+37
```

The source code file for the monitor looks as follows:

*Figure 97. process.csl File*

The monitor uses two arguments, ps_flags which specifies the flags to be used with the ps command and grep_list that specifies what to search for in the ps output. The implementation of the monitor returns an integer value that contains the number of processes found matching the criteria.

To compile the monitor on AIX, use the following command:

```
mcsl -P /usr/ccs/lib/cpp -x process.col process.csl -lang-c++ -nostdinc -undef
```

After stopping and starting oserv, the new monitor will be available.

**Remember**

This monitor example is also in the sample code package for this book. You will find it as one of the options in the Redbook_samples monitoring collection.

See Appendix B, “How to Get the Samples in This Book” on page 141.

## 5.7 Hardware Alerting from AIX Error Report

The AIX operating system has a mighty error logging facility. As a system administrator you should periodically look at the error report to find out if your system has any problems. Ideally you want to be informed automatically and immediately if some really serious conditions, such as hardware problems, occur.

The error reporting mechanism allows for asynchronous alerting. Instead of having a monitor that checks your error report automatically at a scheduled time you can add a new error notification method to the AIX Object Database Manager (ODM) which will be invoked by the error daemon in the event of serious errors and send a notification to a management platform. The following implementation is valid for both Systems Monitor and TME 10 Distributed Monitoring but goes into more detail for TME 10 Distributed Monitoring to show the capabilities of an asynchronous monitor.

In our example we will look for all errors that have the alertable flag set to true. If you are not familiar with the error reporting and/or ODM manipulations you should be especially careful when making modifications and always refer to InfoExplorer, the manual, or the man pages first.

First, you may want to have a look at your existing error log notification methods. Type the following commands:

```
odmget errnotify > /tmp/errnotify_original
vi /tmp/errnotify_original
```

Look at the /tmp/errnotify_original file. Every method filters for one or more of the fields in the error templates, such as en_symptom, en_alertflg or en_label. If the error event fits the requirements, the program specified under an en_method will be run. We will create our own method, triggered by any events having the en_alertflg flag set.

### 5.7.1 Creating the Sentry Asynchronous Monitor Script

Our error notification method will in fact be a simple shell script, /usr/local/bin/errpt/errpt_notification. The error notification interface passes the ID of the actual error report entry to the method.

The /usr/local/bin/errpt/errpt_notification script contains only a few lines:

```
. /etc/Tivoli/setup_env.sh
/usr/local/Tivoli/bin/aix4-r1/bin/wasync -c errpt -s 0 -i "`/usr/bin/errpt -al $1`"
```

The script sets up the Tivoli environment in the first line and then calls an asynchronous monitor, by means of the wasync command. The arguments for the wasync command are:

| | |
|---|---|
| **-c errpt** | Name of the Sentry channel |
| **-s 0** | No trigger data for the monitor |
| **-i '/usr/bin/errpt -al $1'** | The details of the error report entry are passed to the monitor |

Next we need to create a Sentry asynchronous string monitor to catch this wasync request. Figure 98 on page 113 shows how to do this from the GUI.

Inside the monitor we define that at the always response level a pop-up window will open.

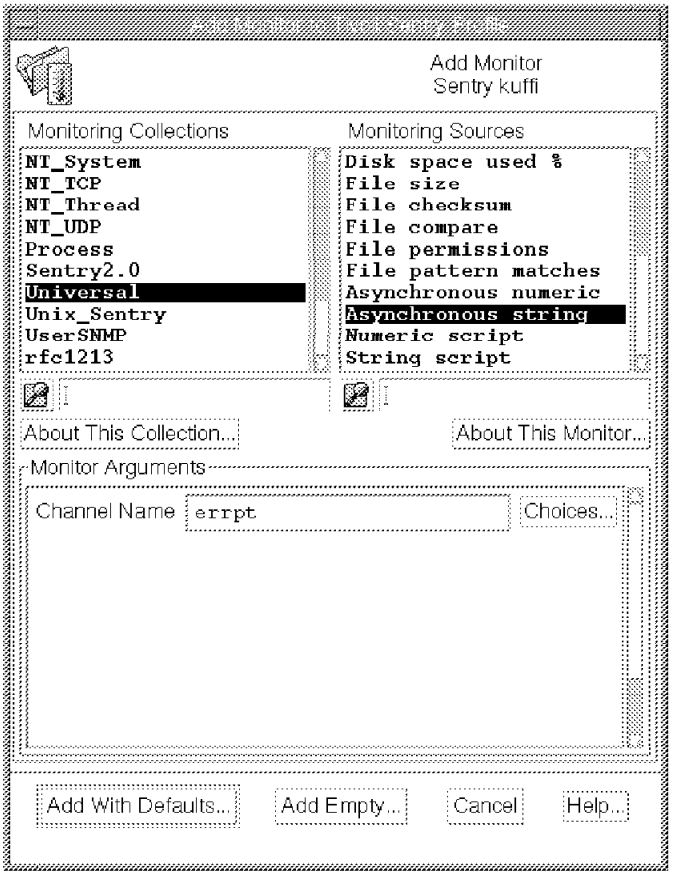



*Figure 98. Definition of the Asynchronous Monitor Using Channel errpt*

## 5.7.2 Installing the Error Notification Method

The notification method is defined as a sequence of attributes listed in an input file, as shown in Figure 99

```
errnotify:
en_pid = 0
en_name = "all_alertable"
en_persistenceflg = 1
en_label = ""
en_crcid = 0
en_class = ""
en_type = ""
en_alertflg = "TRUE"
en_resource = ""
en_rtype = ""
en_rclass = ""
en_symptom = ""
en_method = "/usr/local/errpt/errpt_notification $1"
```

*Figure 99. New Method for AIX Error Notification*

To add this error notification method to the ODM, enter:

```
odmadd <method_definition_file>
```

If you subsequently want to delete the method, enter:

```
odmdelete -o errnotify -q en_name='all_alertable'
```

Of course, you could put in any other command as an en_method. For example, you could send the error message as an SNMP trap to a mid-level manager or to TME 10 NetView itself. Also, you can send an event to T/EC using, for example, the wpostemsg command.

For testing you can create an alertable error manually. To do this you must first temporarily change the error template of the OPMSG (operator message) error type to be an alertable message. The following sequence of commands will do this (¬ D stands for Ctrl+D):

```
errupdate
=AA8AB241:
Alert = True
¬D
¬D
```

In the above example, AA8AB241 is the ID of OPMSG.

Now you can create an alertable error by typing:

```
errlogger "Test for Channel errpt and the Error Notification Method"
```

You should receive a TME 10 Distributed Monitoring alert as shown in Figure 100.

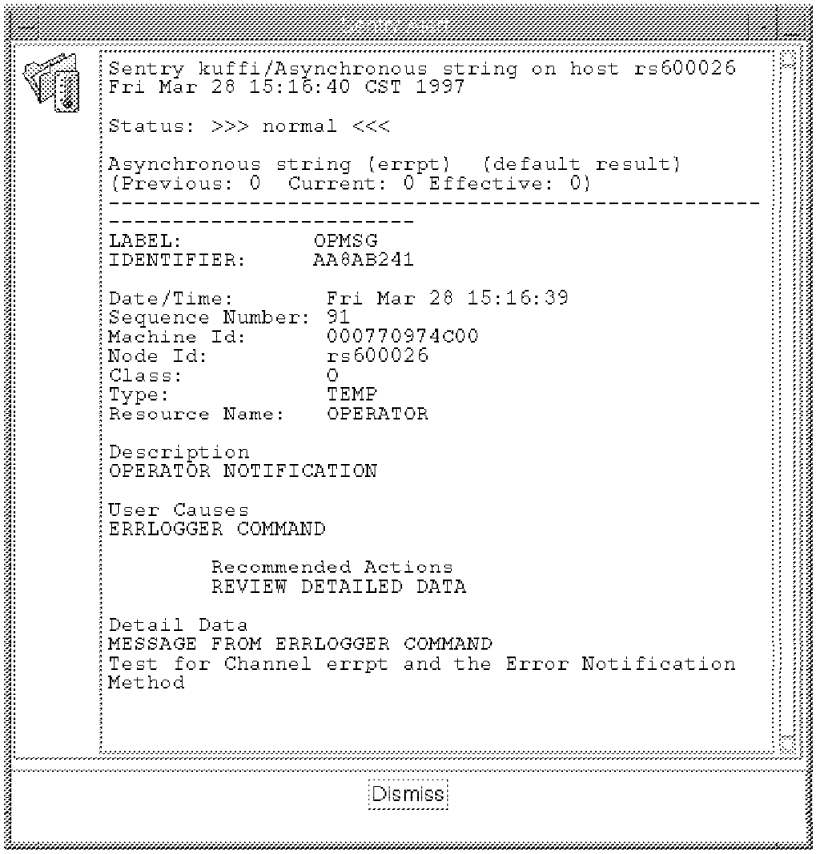



*Figure 100. Sentry Alert Caused by Alertable AIX Error Report Entry*

After you have verified that the error notification works you should reset the template of the operator message back to non-alertable:

```
errupdate
=AA8AB241:
Alert = False
-D
-D
```

## 5.8 Generic File System Monitoring

Generic file system monitoring, as opposed to hard-coding each file system can reduce the maintenance overhead in a large environment. If a new critical file system is added by a user, the administrator normally has to add a monitor and re-distribute the profile every time.

A more efficient method is to monitor all file systems and use an exception list for file systems such as /cdrom or /usr/sys/inst.images that do not need to be monitored.

### 5.8.1 Monitoring with Systems Monitor

Systems Monitor MLM lends itself to this kind of monitoring by allowing for a wildcard definition in the MIB extension. For example, you can create an MLM threshold monitor to poll for the utilization of all file systems by specifying the MIB object for utilization in the SIA file system table and appending an asterisk (*) to signify ″all instances″. The MIB definition is therefore:

.1.3.6.1.4.1.2.6.12.2.5.2.1.4.*.

To perform the exception list processing you need to execute a shell script as the action for this threshold. This script needs to convert the file system name from a dotted-decimal form into plain text. There is an example showing how to do this in *IBM Systems Monitor Anatomy of a Smart Agent,* SG24-4398.

An alert can then be sent to NetView containing the file system name and the percentage full value.

### 5.8.2 Monitoring with and/or Migration to Sentry

To achieve the same effect using TME 10 Distributed Monitoring, a custom monitor is necessary. The custom monitor calls a shell script that executes platform specific probes to check the status of file systems. The sample script presented here has been tested on AIX, DGUX and HP-UX and SunOS.

Figure 101 on page 116 shows the source code of the script. It responds with a message, containing the names and details of all file systems that exceed the utilization threshold value. If all of the file systems are within specification, it simply responds with the text ″OK″. You therefore should use this script with a String script monitor and a response threshold of ″not equal to OK″.

```
#!/bin/ksh
#########################################################################
# FILE:         sentry_filecheck
#
# This script is an example from IBM redbook SG24-4936. It is made freely
# available on the understanding that it is unsupported sample code only
#
# DESCRIPTION:
# This script can be run using the string script monitor. It checks
# the utilization of all filesystems on the monitored system against
# a predefined percentage value and reports any that have exceeded
# it. It can optionally use a control file to exclude specific filesystems
# from being monitored.
#
# ARGUMENTS:
# (1) The utilization threshold limit
# (2) (optional) the name of the exclude file
# (3) (optional) an additilnal list of filesystems to exclude
#
# RETURNED VALUES:
# A string containing the names and utilization of any file systems
# that exceed the limit. If there are no exceptions, the script returns a
# null string
#
# AUTHORS:      Graeme Naysmith IBM UK
#               Rob Macgregor   ITSO Raleigh
#########################################################################
#
# set filesystem threshold and exception file
#
fileused=$1
exception_file="/usr/local/lib/sentry.exception.config"
# Exception file contains a list of blank-separated filesystem names:
# for example: /usr /usr/local/Tivoli

if (( $# > 1 ))
then
 exception_file=$2
fi

if [[ -r $exception_file ]]
then
  cat $exception_file | read except_list
fi

if (( $# > 2 ))
then
 except_list="$except_list $3"
fi

except="egrep -v "
for fs in $except_list
do
  except="$except$fs\$|"
done
except="$except""nothing"

curdate=`date`
system=`uname`
case $system in

  AIX)

    df | $except | awk -v maxsize=$fileused \
    'BEGIN{header="OK"; report=""}
    {if (substr($4,1,length($4)-1)>maxsize || substr($6,1,length($6)-1)>maxsize)
    {
```

*Figure 101. sentry_filecheck Script (Part 1 of 2)*

```
      header="Exception report"
      report=report": "$7" is "$4" full"
    }
    }
    END{print(header report)}' 2> /tmp/fserr
  ;;

 DGUX)

    df -k | $except | awk -v maxsize=$fileused -v date="$curdate " \
    'BEGIN{header="OK"; report=""}
    {if (substr($5,1,length($5)-1)>maxsize || substr($6,1,length($6)-1)>maxsize)
    {
      header="Exception report"
      report=report": "$6" is "$5" full"
    }
    }
    END{print(header report)}
    '}
  ;;


 HP-UX)

    bdf | $except | awk -v maxsize=$fileused -v date="$curdate " \
    'BEGIN{header="OK"; report=""}
    {if (substr($5,1,length($5)-1)>maxsize || substr($6,1,length($6)-1)>maxsize)
    {
      header="Exception report"
      report=report" : "$6" is "$5" full"
    }
    }
    END{print(header report)}
    '}
   ;;

 SunOS)

    df -k | $except | /usr/xpg4/bin/awk -v maxsize=$fileused -v date="$curdate " \
    'BEGIN{header="OK"; report=""}
    {if (substr($5,1,length($5)-1)>maxsize || substr($6,1,length($6)-1)>maxsize)
    {
      header="Exception report"
      report=report": "$6" is "$5" full"
    }
    }
    END{print(header report)}
    '}
  ;;

  *)
    print "# $0 Operating system not recognised"
  ;;

esac
```

*Figure 102. sentry_filecheck Script (Part 2 of 2)*

As we discussed in 5.3.3, “Monitoring UNIX CPU Utilization Using MCSL” on page 81, it is better to place new monitors of this type into a new monitoring collection, instead of requiring the administrator to invoke the shell script from a custom monitor. We added the file system monitoring script to our sample monitoring collection. Figure 103 on page 118 shows how to invoke the monitor, and the resulting event message.

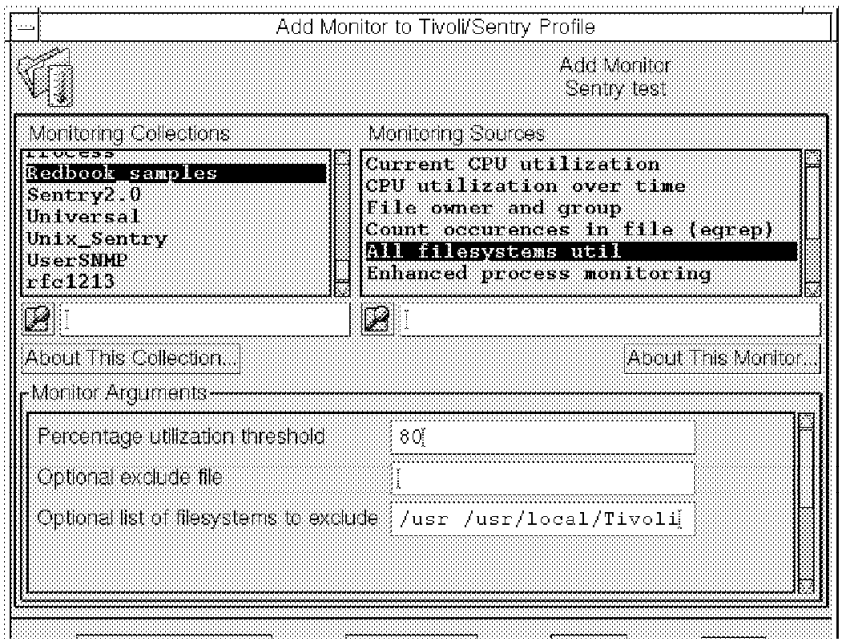



*Figure 103. Using the All Filesystems Monitor*

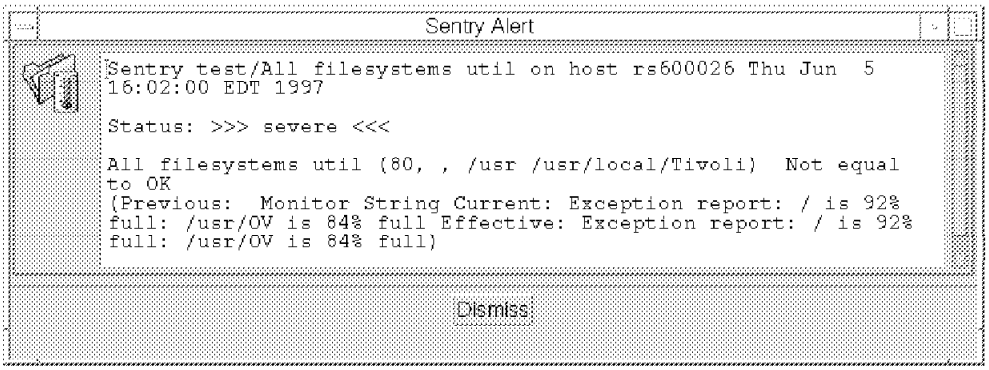



*Figure 104. Alert from All Filesystems Monitor*

## 5.9 SNMP Proxy Forwarding

The TME 10 Distributed Monitoring proxy function allows monitoring of entities that are not directly supported by TME 10 Distributed Monitoring, such as devices or network components not capable of running the TME 10 Distributed Monitoring engine.

The proxy feature is accomplished by running a monitor on a managed node that analyzes the availability of a host, checks log files for messages, checks SNMP values or runs scripts with values set by the proxy's environment variables. The message returned by the managed node running the monitor will be sent on behalf of the endpoint defined in the TME 10 Distributed Monitoring proxy.

### 5.9.1 Monitoring with Systems Monitor

Describing all the SNMP monitoring features of Systems Monitor is beyond the scope of this book. Please refer to *IBM Systems Monitor Anatomy of a Smart Agent*, SG24-4398 for a detailed description of these features.

### 5.9.2 SNMP Monitoring with and/or Migration to Sentry

TME 10 Distributed Monitoring allows you to use SNMP monitoring collections. The following types of MIBs can be monitored:

- Compaq Insight Manager
- MIB-II (RFC 1213)
- User-specified

User-specified means that you can get and compute any simple variable your agent supports. Sentry uses its own SNMP commands, such as wsnmpget to contact the agent. Since right now there is no command like wsnmpnext or wsnmpwalk available, you cannot get SNMP variables of the type table.

Compaq Insight Manager and MIB-II means nothing more than that the two collections have monitors that use a symbolic name for specific MIB variables from particular SNMP MIBs. Figure 105 shows how an rfc1213 monitor looks using the symbolic name and Figure 106 on page 120 shows how the same can be done with a user-specified monitor.

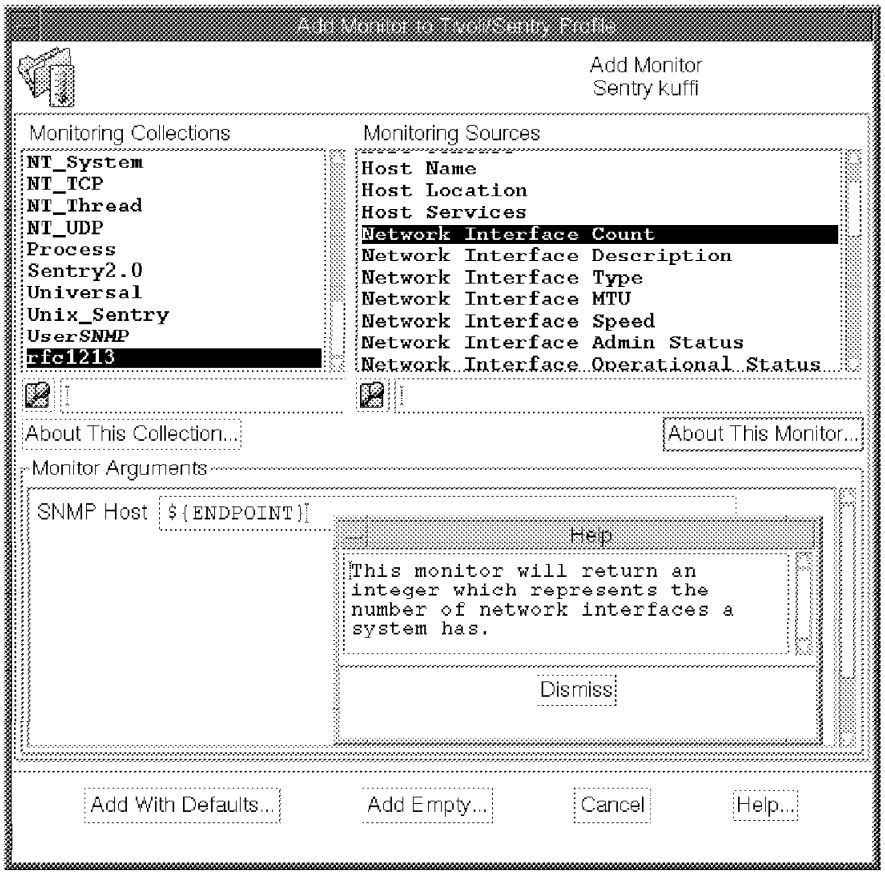



*Figure 105. SNMP Monitor Using Symbolic Names*

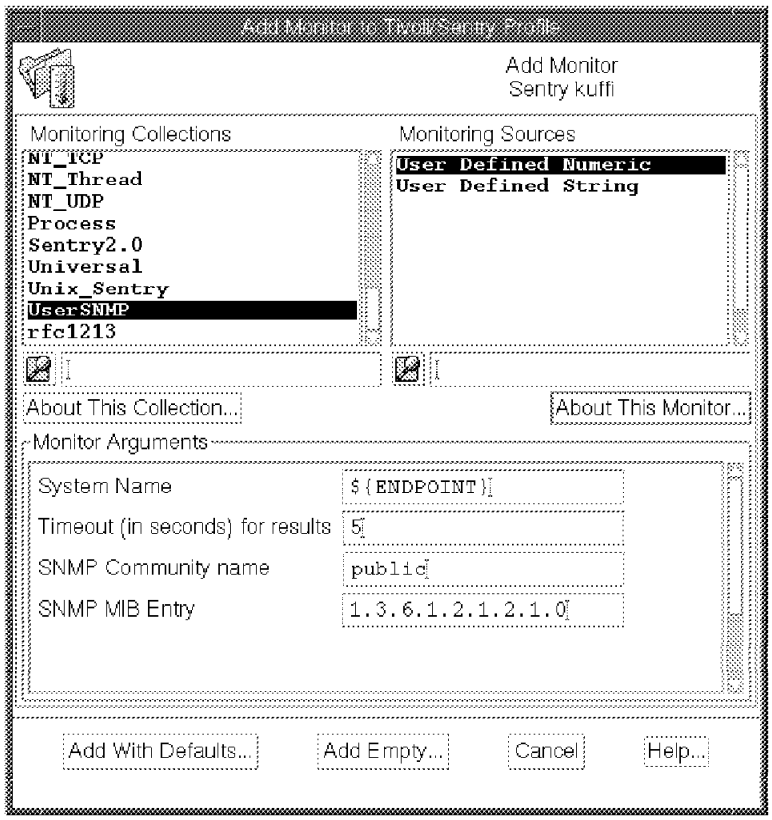



*Figure 106. User-Specified SNMP Monitor*

### 5.9.2.1 SNMP Monitor Example with Proxies and Indicator Collections

TME 10 Distributed Monitoring does not have as many features for monitoring SNMP devices or variables as Systems Monitor, but you still can use it for effective SNMP management. We will use the example of monitoring a user-specified MIB variable, 1.3.6.1.2.1.5.3 (icmp.icmpInDestUnreachs = the number of ICMP Destination Unreachable messages received) at an OS/2 machine.

The MIB variable we are looking at is a good point to look at, for example, if a machine has a corrupted routing table or an application tries to reach a host outside the network. In either case you want to be warned about that fact. Perhaps you would log on to that machine, have a look at the `netstat -rn` output and then figure out a possible reason for the problem.

Every time the client cannot reach a destination for any reason the counter of the MIB variable 1.3.6.1.2.1.5.3 will be increased by one. You can enforce an entry by pinging the non-existing host 1.1.1.1 and since this host is not reachable within your network, an icmp.icmpInDestUnreachs message will be sent increasing the counter by one.

To see this you can type on the system where the proxy resides:

```
wsnmpget -h domingo -t 5 -c public 1.3.6.1.2.1.5.3.0
```

In the above line domingo is the hostname of the SNMP client, 5 is the time-out value, public is the SNMP community and 1.3.6.1.2.1.5.3.0 is the SNMP object.

Now try to ping the bogus machine:

```
ping 1.1.1.1
```

Wait a few seconds and then query the variable again by repeating the wsnmpget command.

### 5.9.2.2 Setting Up a Sentry Proxy Endpoint

Every client you want to run your monitors on has to be defined in the Tivoli database. To achieve this we have to create a proxy endpoint. We can do this by selecting **Create** from the menu bar in the Policy Region window and then selecting **Sentry Proxy...** from the pull-down menu. The window shown in Figure 107 will appear.

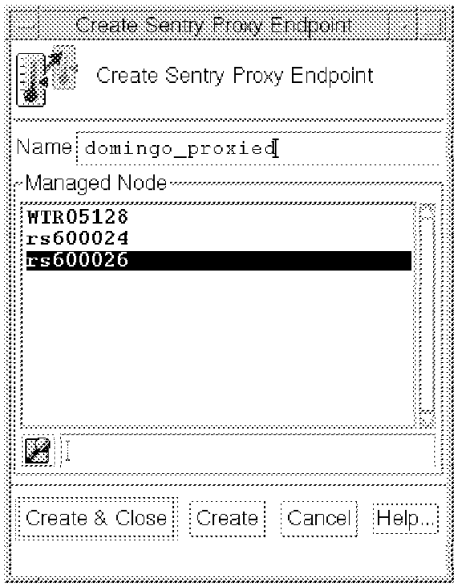



*Figure 107. Creating a Sentry Proxy Endpoint*

We enter a name for the new Sentry proxy endpoint and then select the **Create & Close** button.

This creates an icon in the policy region representing the new proxy endpoint. Double-clicking on that icon will open the Proxy Endpoint window. In this window we select **Configure** from the menu bar and then **Set Environment...** from the pull-down menu. The display shown in Figure 108 on page 122 will appear:

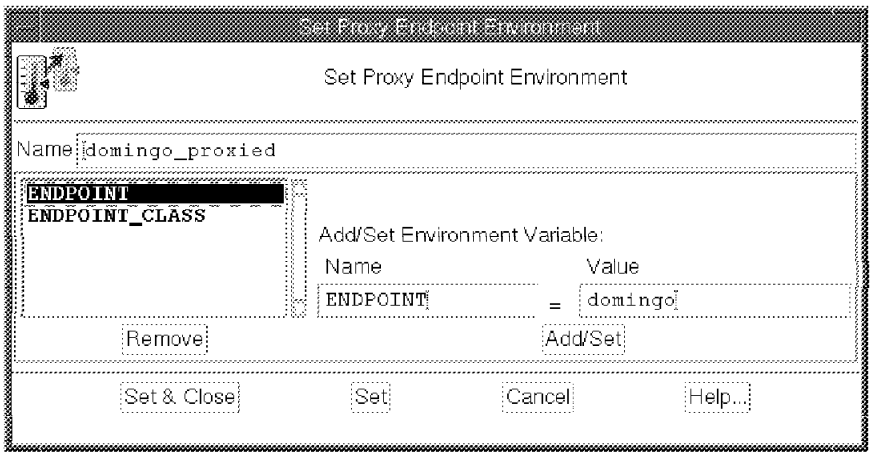



*Figure 108. Setting the Environment for Sentry Proxy Endpoint*

We enter ENDPOINT in the Name field and domingo in the Value field and then press the **Add/Set** button to set the ENDPOINT variable. After that we enter ENDPOINT_CLASS in the Name field and SentryProxy in the Value field and press the **Add/Set** button again.

ENDPOINT contains the hostname of the system to be monitored and ENDPOINT_CLASS is the Tivoli class name of the endpoint object.

We would need to repeat this step for each client we want to include. Of course, in a larger environment you would use the command line interface.

To create the proxy endpoint you can type:

```
wcrtprx region proxy endpoint_name
```

In our example:

```
wcrtprx ssd-region WTR05128 domingo_proxied
```

To set the environment variables you can type:

```
wcrtprxenv domingo_proxied 'ENDPOINT_CLASS=SentryProxy' 'ENDPOINT=domingo'
```

### 5.9.2.3 Creating an Indicator Collection

Creating an indicator collection is a very easy step. In the Policy Region window select **Create** from the menu bar and then **IndicatorCollection...** from the pull-down menu. Give the indicator collection a name (we use SNMP_proxied_Dest_unreachable) and select **Create & Close**. This will create an icon in your policy region representing the new indicator collection.

### 5.9.2.4 Creating a User-Specified SNMP Monitor

We already showed in 5.9.2, “SNMP Monitoring with and/or Migration to Sentry” on page 119 how to set up a user-specific SNMP monitor. Just update the MIB variable to `1.3.6.1.2.1.5.3.0`.

Do not forget to tell the profile manager, in which the profile resides, that it has some new subscribers. In fact, all the proxy endpoints created before should subscribe to the same profile manager. Look at Figure 109 on page 123 to see how we set up the triggering. Every time the value of the SNMP variable

increases by 3 or more, the indicator collection will be notified. This is specified by checking **Change Icon**.

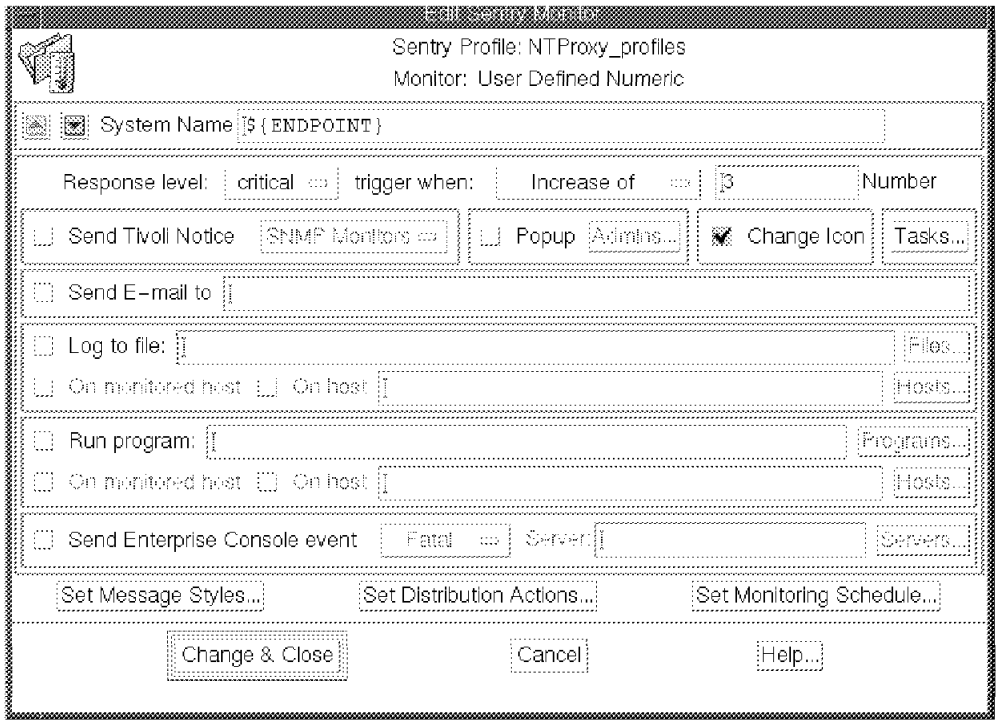



*Figure 109. Sentry Profile for Proxy Monitoring*

Finally, we have to tell the monitor with which indicator collection it is connected. In the Profile Properties window of the Sentry profile select **Monitoring** from the menu bar and then Select **Indicator Collection...** from the pull-down menu. Then select the indicator collection you have created before from the list and press the **Select & Close** button.

Now distribute the profile again and double-click at the proxy endpoint. You should get a window as shown below:

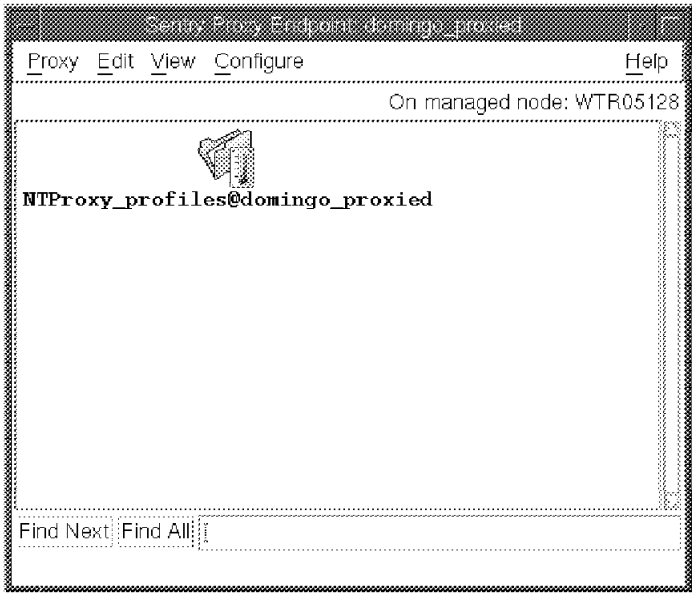



*Figure 110. Sentry Proxy Endpoint WIndow*

Once your monitor is running you can look at the indicator (Figure 111) to see if something is wrong with this particular SNMP variable on your proxy clients.

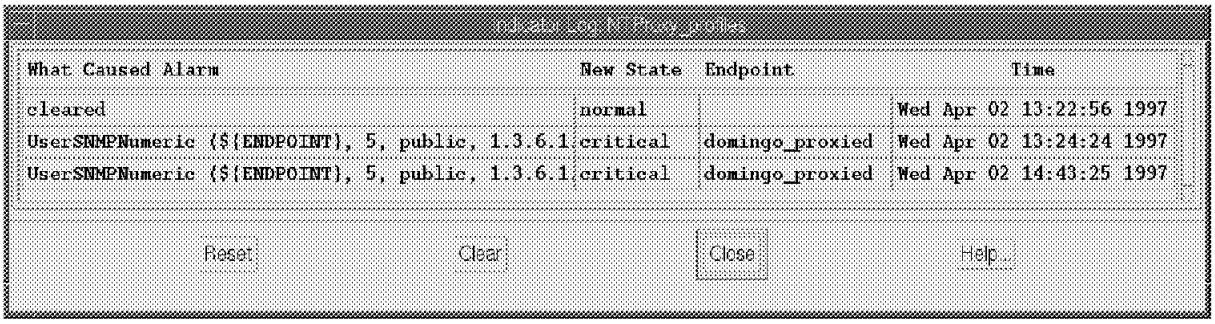



*Figure 111. Indicator Collection Messages*

## 5.10 Migrating Data Collection

The MLM and SLM threshold table has a data collection function as well as the threshold monitoring capability that we have been examining so far. Prior to TME 10 Distributed Monitoring 3.5, there was no equivalent function as standard in Sentry, although it is possible to quite simply create a monitor that writes data to a log file.

TME 10 Distributed Monitoring 3.5 introduces a graphical reporting capability that we briefly described in 5.3.5, “Installing the UnixCPU Collection on Another TMR Server” on page 91. This function is made up of a data capture capability that logs the result of normal monitors to a file and 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 that will read the TME 10 Distributed Monitoring log files and summarize the figures in its database.

The graphical data collection facility is more fullly described in *A First Look at TME 10 Distributed Monitoring 3.5*, SG24-2112.

### 5.10.1 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.

### 5.10.2 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 112 shows an example of doing this for a monitor that records the free space in a UNIX file system.

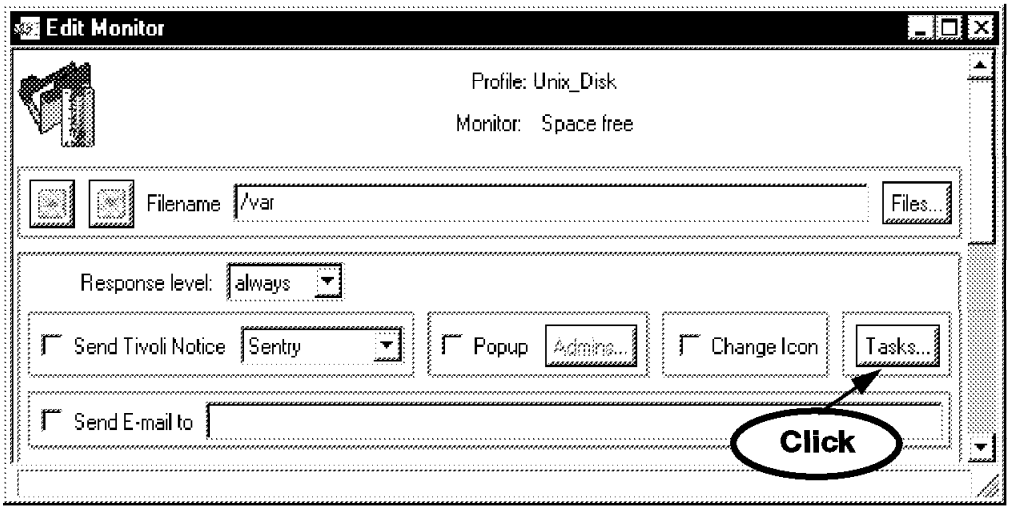



*Figure 112. Defining an Always Response*

In the Tasks dialog, select **Sentry Graphable Logs** as the task library and select task **Create Graphable Log** (see Figure 113 on page 126).

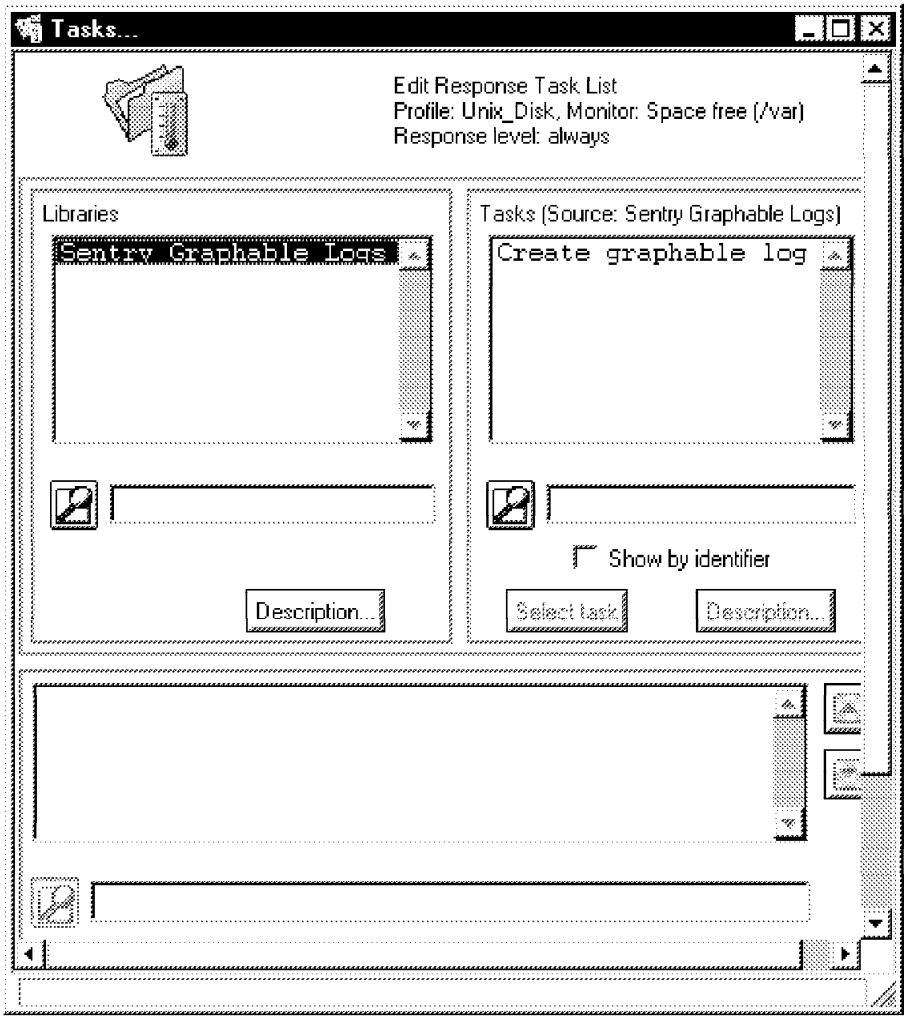



*Figure 113. Defining the Logging Task*

When you define the Create 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 plenty big enough. For example, if you are collecting at 10-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* that contains the data itself.

### 5.10.3 Extracting Logged Data from the Command Line

We described how the data collection task writes its log files in 5.10.2, "Collecting Monitor Data" on page 125. 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 timestamp, 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 applications 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 monitor

As an example, Figure 114 on page 128 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 114. Examples of the wgdread Command*

The timestamps 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 115 shows a simple perl program that can be used as a filter for the output of wgdread and Figure 116 on page 129 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 115. 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 116. Extracting Historical Data with Date and Time Conversion*

# Chapter 6. Installation Notes and Trouble Shooting

During the project we set up a number of different TME configurations spanning several different operating systems and network environments. We will not describe the installation and customization process in any detail here, but we will briefly discuss some of the problems we encountered, and their resolutions.

## 6.1 Installation Notes

We used the following monitoring products for our testing:

- Systems Monitor SIA 2.3.1.0
- Systems Monitor MLM 2.3.1.0
- Systems Monitor CFG 2.3.1.0
- Tivoli/Sentry 3.0.2
- Tivoli/Sentry 3.5

Also, since TME 10 Distributed Monitoring is a TME 10 Framework application we had to install the TME 10 Framework and organize the machines in our setup into Tivoli Management Regions (TMRs). We used the following levels of Framework and support code:

- TME 10 Framework 3.1 Revision C with patch 0002 installed
- TME 10 ADE 3

Because Systems Monitor is primarily an AIX application, most of our test machines were RS/6000s, running AIX 4.1.4, 4.1.5 and 4.2. We also had an HP9000 running HP-UX 9.0.5 and a Sun Sparc 20 running Solaris 2.5, plus a number of Windows NT and OS/2 systems.

### 6.1.1 Installation of the TME 10 Framework

For a detailed description of how to install and configure the TME 10 Framework refer to the redbook *TME 10 Cookbook for AIX*, SG24-4867. On each system we performed a number of additional setup steps:

- Added .profile for the root user ID, invoking the Tivoli environment setup script, as follows:

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

- Added /usr/local/bin to the default path in /etc/environment.
- Created file system /var/spool/Tivoli with 80 MB on the TMR server, 20 MB on other managed nodes.
- Created file system /usr/local/Tivoli with approximately 100 MB (250 MB on the T/EC server).

### 6.1.2 Notes on Backup and Restore

We set up a daily scheduled TME backup for each of our TMRs. It is important to check that the backups do not fill the /var/spool/Tivoli ($DBDIR) file system; otherwise, you may find yourself with a number of useless backup files. The backup process uses a lot of space in the file system while it is processing and then frees it later. This means that a backup or restore may fail, even though the file system has apparently not run out of space. The symptom of this is a sequence of messages like the following:

```
-> no format string
-> Date, Time, (18): 'iom_open' failed with code '67':
-> *no format string*
```

We used the backup on a number of occasions to re-create a configuration for testing purposes. In order to perform backup and restore operations your administrator ID needs to have the appropriate authorization. To do this, open the **Administrators** window from the TME desktop, click on the administrator icon with the right mouse button and select **TMR roles**. Then select **restore** and **backup** from the list.

It's always a good idea to check the database integrity before making a backup. The command is: wchkdb -ux.

### 6.1.3 TME 10 NetView Installation

We created a /usr/OV file system of about 200 MB, NetView Base took about 160 MB of this space. We were using a beta version of TME 10 NetView V5, which may have been larger than a final version of the code would be.

### 6.1.4 Systems Monitor Setup

We had some problems with the ″command to run before monitor″ function not working correctly on Systems Monitor SIA. The solution to this was to install PTF U446881. We also installed PTF U444055 on the MLM and U444057 on the configuration EUI components.

Because Systems Monitor uses SNMP for remote agent configuration, you have to define an SNMP community name with write authority. This involves adding an extra community definition in /etc/snmpd.conf on the managed node, and defining the same community name in the TME 10 NetView SNMP configuration dialog.

## 6.2 Problems and Resolutions

In this section we describe some problems that we encountered during the project and the circumvention or fix that we used to resolve them.

### 6.2.1 TME Installation Revision Levels

Sometimes when installing a Sentry product for the first time on a managed node, the install was rejected at the point at which it verifies prerequisites. The message it gives is not too helpful:

```
The revision level of the installation and media do not match
″Sentry 3.0.1>3.0″
```

What this means is that you have installed a product on the TMR server and then applied patches to bring it to a higher level (in this case, 3.0.1). Now you are installing the base on a managed node in the TMR, but Sentry requires that the server's and client's code levels match and therefore kills the request. This problem is not limited to Sentry, and there are a number of different ways to get a similar problem, depending on the order in which you install the distributed systems.

The fix is to fool the install process into thinking that the levels match. First enter the following on the server:

```
idlcall `wlookup -r ProductInfo Sentry2.0.2` _get_revision
```

In the case of the error message shown above, the response from this should be ″3.0″. Next update the apparent revision level:

```
idlcall `wlookup -r ProductInfo Sentry2.0.2` _set_revision '″3.0″'
```

You should check that this worked using the same display command as before:

```
idlcall `wlookup -r ProductInfo Sentry2.0.2` _get_revision
```

Now you should be able to continue with the installation.

## 6.2.2 Sentry Engine Not Running

Once or twice we found that monitors were not working because the Sentry engine had stopped running on the system (either due to a failure or because we had manually stopped it). This presents a dilemma, because although there is a command to stop the engine, the wstopeng command, there is no equivalent wstarteng command. Normally the engine is started automatically when the object dispatcher, oserv, starts.

The solution to this is simply to query the engine status:

```
wlseng <hostname>
```

The engine should now restart.

## 6.2.3 Sentry Monitors Not Working

There are many reasons why Sentry monitors may not work as you expect them to, but the following sequence of checks should help you track down the problem.

### 6.2.3.1 Check Monitor Installation

First, make sure that the monitor has been installed properly on the managed node and is scheduled to run. The command to do this is:

```
wlseng -l <hostname>
```

This shows you the monitors that the engine knows about, when they are next expected to run, and what the last response was. If you do not see your monitor here, check that it is not disabled (the command does not list disabled monitors, unless you add the -d flag).

#### 6.2.3.2 Check Response Is What You Expect

If the monitor is listed, the next thing to do is to set it to log to a file, so that you can check the response. It may be that the monitor is working but the situation in which it would trigger a response is not occurring. The way to do this is to set the ″Always″ response level on the monitor to log to a file.

#### 6.2.3.3 Check Permissions

On a number of occasions we experienced problems with monitors because they needed authorization to resources. This could be either a UNIX permissions problem or a problem of not having the necessary TME administrator authority. It can arise both in the monitor itself and also in automated actions on a response level.

The Sentry engine runs monitors under user ID nobody in group nobody by default. You can check whether your monitor will operate in this regime by switching to user ID nobody and manually running it (use the command `su - nobody` to switch IDs). You can set the ID under which the monitor runs by opening the user profile window and selecting **Edit** followed by **Set User & Group ID** from the menu bar (see Figure 58 on page 71). You can check this setting with the command:

```
wdumpstr <SentryProfile>
```

If your monitor needs to issue Tivoli commands (such as the example in 5.5.2, “A Closer Approximation to Re-Arm Using Sentry” on page 104) it will require a TME administrator ID. This means that the user ID under which the monitor runs must map to an administrator ID with sufficient authority to do the job.

#### 6.2.3.4 Custom Script Monitors Not Working

The following checklist may help if you have created a custom script that does not seem to work properly:

- Does the script print its result to stdout?
- Does it exit with return code zero (″exit 0″ in Korn shell script)?
- Does it set the execution environment? A monitor script must set the shell to execute under in its first line (that is, ″#!/bin/sh″, or equivalent).
- Is the shell script executable by the user ID that the monitor is running under (default: nobody)?

One debug technique that we used was to modify the script to execute the `env` command to a log file, to see if it had the environment it needed.

### 6.2.4 Other Problems

Problems beyond the errors described above are usually either related to load (trying to make the Sentry engine do too much) or are caused by code bugs. In the latter case, you should report them to Tivoli, but the techniques described below may get you working again while waiting for a permanent fix.

On the question of load, you should keep in mind two things:

1. TME 10 Distributed Monitoring performs most of its work on the monitored system. Be careful not to introduce monitors or actions that invoke TME services on every monitor cycle; otherwise, you will add a lot of traffic to the network and the server. Any Tivoli CLI command requires at least two calls to the server to authenticate and check authorization.

2. Sentry engine works on one-minute boundaries for its monitoring processes. Within each one-minute slot, it checks for monitors that are scheduled to run and then runs them sequentially. This means that if you have a monitor that takes an exceptionally long time to complete, or too many monitors scheduled in the same time slot, you can overload the engine. Try to avoid setting monitor intervals too short and beware of commands that can take a long time to run.

We will now look at two specific problem symptoms and how to circumvent them.

#### 6.2.4.1 Sentry Engine Internally Blocked

If you suspect that Sentry is not working and run the wlsmon command, you may see a message that the engine is internally blocked. This usually means that it has become overloaded and cannot process monitors or commands within the current time slot. You should wait a short time and retry the command, but if the problem that caused it to get blocked persists, the message will recur. The long-term solution is to reduce the load on the engine by cutting back the number and frequency of monitors and by identifying and fixing any monitors that may be taking a long time to run. However, to get the Sentry engine running again in the short term, you need to clear its queue of monitors to run and re-initialize it. The script shown in Figure 117 will clean up the engine. Redistribute monitors after running it.

```
#!/bin/sh
#####################################################
#
#
# script to clear "sentry engine internally blocked" problem
#
#
# using commands supplied by Sean Starke at Tivoli
#
#####################################################

if [ $# -ne 1 ]
then
  print "Please input hostname"
  exit
fi

. /etc/Tivoli/setup_env.sh

wlseng
print "Clearing Sentry engine"

oid=`wlookup -r ManagedNode -n $1 SentryEngine`
print $oid

idlattr -t -s "$oid" sentry_engine_state imp_SentryEngine::EngineState '{0 0 0 ""}'

#$BINDIR/TME/SENTRY/wclreng $1
/usr/local/Tivoli/bin/aix4-r1/bin/wclreng $1

wlseng

exit 0
```

*Figure 117. sentry_cleanup Script*

#### 6.2.4.2 General Failure Message

Some Sentry problems are related to more serious oserv failures. In these cases, the wclreng command may help to reset the monitoring processes. After running it, resynchronize with the server by redistributing the Sentry profiles with the exact copy option.

# Appendix A. Mapping SIA MIB Objects to Sentry Monitoring Collections

Systems Monitor SIA provides a great deal of information about the system it is running on. By comparison, TME 10 Distributed Monitoring has a much smaller number of available monitors. Fortunately, a simple comparison of the number of monitors is not the whole story. Many of the SIA system metrics are of limited value, so you are unlikely to find them used in your configuration. On the other hand, although there are far fewer Sentry monitors, they tend to be of more practical use.

The SIA monitors are divided into a series of MIB tables. The tables and the objects in them are identified by a name and by a MIB object ID (OID). The equivalent Sentry monitor is listed beside it, plus some comments if there are detail differences between the way the two values are calculated. *If a Systems Monitor MIB variable is not listed, there is no equivalent Sentry Monitor*.

All of the monitors listed are from the Unix_Sentry monitoring collection, although several of them are also found in the Universal monitoring collection.

*Table 6 (Page 1 of 3). List of Unix_Sentry Monitors*

| SIA MIB Identifier | Sentry Monitor | Comments |
|---|---|---|
| smSiaSystemDeviceEthernetMaxCollision .1.3.6.1.4.1.2.6.12.2.3.3.2.1.36 | Network collisions | Not an exact match, because the SIA MIB object is part of of a table, with one row per Ethernet interface, whereas the the Sentry monitor is a sum taken over all interfaces. For a system with one Ethernet interface they should be equivalent. |
| smSiaSystemDeviceTokenRingRxErrCnt .1.3.6.1.4.1.2.6.12.2.3.2.2.1.15 **OR** smSiaSystemDeviceEthernetRxErrCnt .1.3.6.1.4.1.2.6.12.2.3.3.2.1.15 | Input packet errors | The SIA MIB objects are part of a table, with one row per network interface. The Sentry monitor is a sum taken over all interfaces (of any type). On a system with one active network interface they should be equivalent. |
| smSiaSystemDeviceTokenRingTxErrCnt .1.3.6.1.4.1.2.6.12.2.3.2.2.1.14 **OR** smSiaSystemDeviceEthernetTxErrCnt .1.3.6.1.4.2.6.12.2.3.3.2.1.14 | Output packet errors | As above |
| smSiaSystemDeviceTokenRingPktTxCnt .1.3.6.1.4.1.2.6.12.2.3.2.2.31 | Output packets | The SIA MIB object is part of a table, so the same comments as above apply. |
| smSiaSystemFreePagingSpace .1.3.6.1.4.1.2.6.12.2.4.1 | Available swap space | Identical |
| smSiaSystemPagingSpaceUsed | Available swap space | In fact this is not an equivalent monitor, but it is often used to track paging space utilization. It is part of a table, with one entry per paging space, so it may not be a good value to threshold on (one paging space could be much fuller then another). Consider using the available swap space monitor as a replacement. |

*Table 6 (Page 2 of 3). List of Unix_Sentry Monitors*

| SIA MIB Identifier | Sentry Monitor | Comments |
|---|---|---|
| smSiaSystemPagingStatisticsPagesPagedOut .1.3.6.1.4.1.2.6.12.2.4.5.1.7 | Page-outs | The SIA MIB object is part of a table. It has a sampling interval controlled by the setting of another MIB object, smSiaSystemPagingStatisticsPollingInterval. By contrast, the Sentry monitor has a fixed sampling interval of 5 seconds. If you set the SIA sampling interval to 5 seconds, the two monitors should return equivalent results, except that the SIA version is in number of pages and the Sentry version is in kilobytes. |
| smSiaSystemFilesystemFree .1.3.6.1.4.1.2.6.12.2.5.2.1.3 | Space free | SIA table has one row per file system, using the file system name as an index. Sentry monitor requires the file system name as an argument. |
| smSiaSystemFileSystemPercentUsed .1.3.6.1.4.1.2.6.12.2.5.2.1.4 | Percent space used | As above |
| smSiaSystemFileSystemInodesUsed .1.3.6.1.4.1.2.6.12.2.5.2.1.5 | Inodes used | As above |
| smSiaSystemFileSystemInodesPercentUsed .1.3.6.1.4.1.2.6.12.2.5.2.1.6 | Percent inodes used | As above |
| smSiaSystemProcessorPID .1.3.6.1.4.1.2.6.12.2.7.2.1.2 | Daemon status | This is not really a direct match, but it can be used for similar purposes. smSiaSystemProcess is a multi-line table with one row per active process. It contains information about the process, including details like the process ID (PID) as in this case. This is often used with an **exists** test to see if a given daemon is running. The Sentry daemon status monitor does a similar job. |
| smSiaSystemProcessState .1.3.6.1.4.1.2.6.12.2.7.2.1.11 | Lingering terminated processes | The process state field is part of a table with one row per process. It returns an enumerated value that indicates the state of the process. A value of 5 indicates a zombie process, that is, one that has terminated but has not yet been acknowledged by its parent. The Sentry monitor returns a count of all such processes on the system. |
| smSiaSystemUsersLoggedin .1.3.6.1.4.1.2.6.12.2.8.1 | Users logged in | Identical |
| smSiaCommandDisplayStringResult .1.3.6.1.4.1.2.6.12.4.1.1.13 | String script | The SIA command table can execute any command, whereas the Sentry monitor always requires a script to execute. However, the same function is obtainable by wrapping the command within a script. You will also have to check the SIA command for use of Systems Monitor provided environment variables. |

*Table 6 (Page 3 of 3). List of Unix_Sentry Monitors*

| SIA MIB Identifier | Sentry Monitor | Comments |
|---|---|---|
| smSiaCommandIntegerResult<br>.1.3.6.1.4.1.2.6.12.4.1.1.14<br>**OR**<br>smSiaCommandCounterResult<br>.1.3.6.1.4.1.2.6.12.4.1.1.15<br>**OR**<br>smSiaCommandGaugeResult<br>.1.3.6.1.4.1.2.6.12.4.1.1.16 | Numeric script | As above |

# Appendix B. How to Get the Samples in This Book

You can download the code for all of the examples in this book from the Web. Go to http://www.redbooks.ibm.com and click on the **Downloads** link. You will find the code package identified by the publication number, SG24-4936.

The package is a tar archive. Download it into a temporary directory on AIX and unpack it with the following command:

```
tar -xvf ./sg244936.tar
```

This will create three subdirectories:

- migration_scripts contains the chk_mig_cat and migrate_sysmon_config tools to help you migrate Systems Monitor configurations to TME10 Distributed Monitoring.
- monitor_collection contains the redbook samples monitoring collection, which incorporates all of the custom monitors that we used in our examples, such as the UNIX CPU monitor, all file system monitors and several others.
- miscellany contains all of the other scripts that do not fall into the other categories.

# Appendix C. Special Notices

This publication is intended to help systems specialists, planners and administrators to migrate from Systems Monitor for AIX to TME 10 Distributed Monitoring. 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 Monitoring 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:

| | |
|---|---|
| ADSTAR® | AIX® |
| IBM® | InfoExplorer™ |
| NetView® | OS/2® |
| RS/6000™ | SystemView® |

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, Windows NT and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Other company, product, and service names may be trademarks or service marks of others.

| | |
|---|---|
| DCE | The Open Software Foundation |
| Domino, Lotus Notes, Lotus, LotusScript, Notes, Replication, NotesView | Lotus Development Corporation |
| Netscape, Netscape Navigator | Netscape Communications Corporation |
| OpenView | Hewlett-Packard Company |
| Tivoli, Tivoli Management Environment, TME 10 | Tivoli Systems Inc., an IBM Company |
| IPX | Novell, Incorporated |
| Intel | Intel Corporation |

# Appendix D. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

## D.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How To Get ITSO Redbooks” on page 147.

- *Systems Monitor for AIX: Anatomy of a Smart Agent,* SG24-4398
- *TME 10 Cookbook for AIX*, SG24-4867
- *The TME 10 Deployment Cookbook: Courier and Friends*, SG24-4976
- *A First Look at TME 10 Distributed Monitoring 3.5*, SG24-2112

## D.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

| CD-ROM Title | Subscription Number | Collection Kit Number |
|---|---|---|
| System/390 Redbooks Collection | SBOF-7201 | SK2T-2177 |
| Networking and Systems Management Redbooks Collection | SBOF-7370 | SK2T-6022 |
| Transaction Processing and Data Management Redbook | SBOF-7240 | SK2T-8038 |
| 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 |

## D.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 lists of redbooks:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

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

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.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.

## How Customers Can Get ITSO Redbooks

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

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

| | 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 Notrh America | (long distance charges apply) |
| (+45) 4810-1320 - Danish | (+45) 4810-1020 - German |
| (+45) 4810-1420 - Dutch | (+45) 4810-1620 - Italian |
| (+45) 4810-1540 - English | (+45) 4810-1270 - Norwegian |
| (+45) 4810-1670 - Finnish | (+45) 4810-1120 - Spanish |
| (+45) 4810-1220 - French | (+45) 4810-1170 - Swedish |

- **Mail Orders** — send orders to:

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

- **Fax** — send orders to:

| | |
|---|---|
| United States (toll free) | 1-800-445-9269 |
| Canada | 1-800-267-4455 |
| Outside Notrh America | (+45) 48 14 2207 (long distance charge) |

- **1-800-IBM-4FAX (United States)** or **(+1) 408 256 5422 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **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).

## IBM Redbook Order Form

**Please send me the following:**

| Title | Order Number | Quantity |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

| | | |
|---|---|---|
| First name | Last name | |
| Company | | |
| Address | | |
| City | Postal code | Country |
| Telephone number | Telefax number | VAT number |

☐ Invoice to customer number _____

☐ Credit card number _____

| | | |
|---|---|---|
| Credit card expiration date | Card issued to | Signature |

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

**DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.**

# Index

## Special Characters



## Numerics



## A



## B



## C

## D



## E



## F



## G

## N



## O



## P



## Q



## R

## S



## T

## U



## V

## W

# ITSO Redbook Evaluation

Migrating from Systems Monitor for AIX to TME 10 Distributed Monitoring
SG24-4936-00

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.