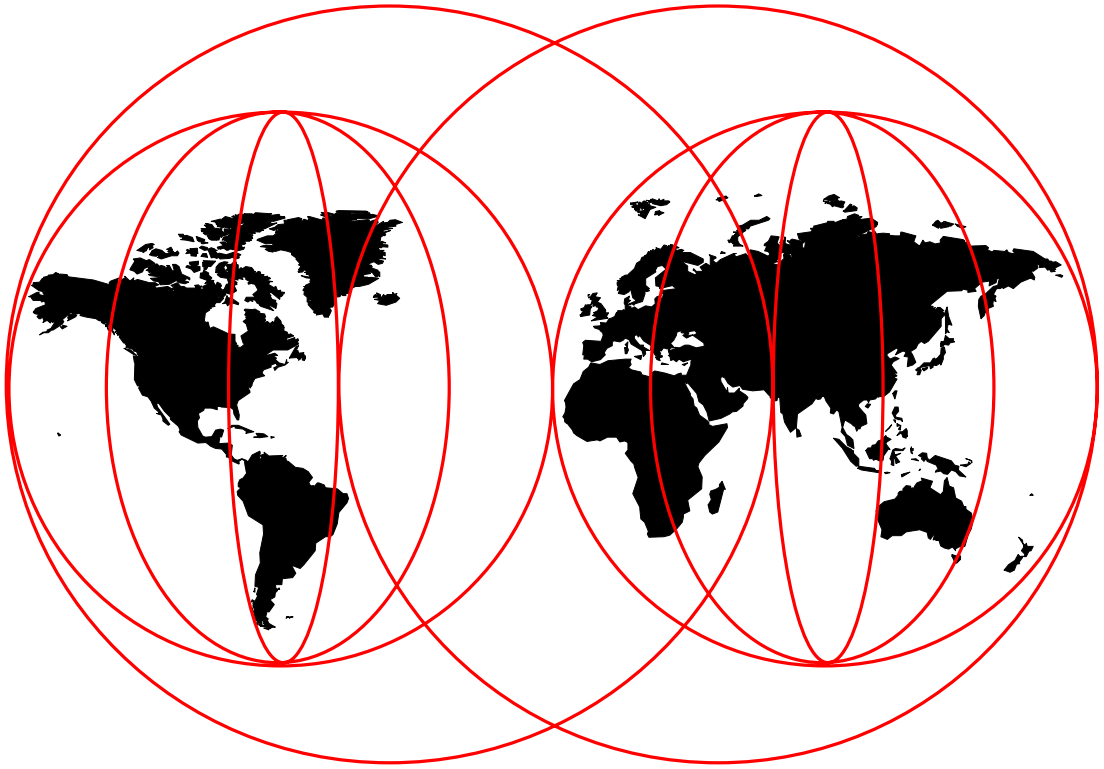
IBM

Creating Custom Monitors for Tivoli Distributed Monitoring

Stefan Uelpenich, Robi Banerjee, Peter Holm, Alain Queffelec



International Technical Support Organization

http://www.redbooks.ibm.com

Tivoli

SG24-5211-00

International Technical Support Organization

SG24-5211-00

Creating Custom Monitors for Tivoli Distributed Monitoring

May 1998

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix A, “Special Notices” on page 219.

First Edition (May 1998)

This edition applies to Version 3.6 of Tivoli Distributed Monitoring, 5697-EMN for use with the AIX, HP-UX, Solaris, Windows 95 and Windows NT operating systems.

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

In this redbook we describe how to create custom monitors for Tivoli Distributed Monitoring, Tivoli's application for monitoring resource availability in the enterprise.

We explain the different approaches you can employ in creating your own monitors and also show specific examples for custom monitors that we create to extend the reach of Tivoli Distributed Monitoring.

We also give a preview of the newest features added in Tivoli Distributed Monitoring 3.6, in particular support of the Lightweight Client Framework (LCF).

This redbook will help you position Tivoli Distributed Monitoring and TME 10 as applications for enterprise availability management and enterprise systems management.

Numerous practical examples are used to illustrate how to use Tivoli Distributed Monitoring in an effective way and how to create your own monitors.

We show how to create monitors for networks, networking hardware, the Web, VM, DB2 and a number of other components and employ different techniques of creating monitors in these examples.

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.

Stefan Uelpenich is an Advisory ITSO Representative, working as a project leader at the Systems Management and Networking ITSO Center, Raleigh. He applies his extensive field experience as an I/T architect and project leader to his work at the ITSO, where he writes extensively and consults worldwide on all areas of systems management. Before joining the ITSO, Stefan worked in IBM Germany's Professional Services organization as an Advisory I/T Architect for Systems Management, consulting major IBM customers. In this role, he architected the configuration management solution for one of Germany's largest client/server networks. Stefan has published a number of books on several areas of Tivoli and systems management.

Robi Banerjee is an Advisory Software Engineer with IBM Global Services in Tampa, Florida. He has been with IBM for three years and has experience in UNIX systems programming and developing network management tools using C/C++ and Perl. He is currently responsible for development and support of ServerD, which is a tool used in Network Services for monitoring servers on different UNIX platforms and Windows NT. Robi holds a Masters degree in Computer Science from Brooklyn Polytechnic University, New York. Prior to joining IBM he worked as a software developer and a quality assurance analyst.

Peter Holm is a Senior Advisory Systems Management Specialist, working in the IBM Nordic PSS Systems Management team, located in Sweden. The team

provides support for Tivoli products for the IBM Nordic region. He joined IBM in 1977 and has worked with network and systems management in technical support, development, and marketing. Before his current position he provided technical marketing support for systems management products at the IBM Nordic Lab in Sweden, and also represented IBM at several trade shows like CMG, UKCMG, and AFCOM.

Alain Queffelec is a Software Engineer, working in Product Support Services in IBM France, located in Paris. His team provides support for Tivoli products in France. Alain joined IBM in 1989 to work in Manufacturing and Development technical services. He has written extensively on test programs for electronics components. He has been working in his current position since 1995 to support the AIX operating system, and Tivoli products since 1997. His areas of expertise include UNIX system administration a network management.

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

Paul Fearn
Nancy Lewis
Linda Robinson
Shawn Walsh
Gail Wojton
Systems Management and Networking ITSO Center, Raleigh.

Dave Hart
Carey Jung
Spike White
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 229 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

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

- Send us a note at the following address:

  `redbook@us.ibm.com`

Chapter 1. Introduction

In this chapter we give a brief overview of Tivoli Distributed Monitoring and the creation of custom monitors.

We explain TME 10, Tivoli Distributed Monitoring, the Lightweight Client Framework and other related matters.

We also give an overview of the new features in Tivoli Distributed Monitoring 3.6.

Disclaimer

This redbook is not meant as a detailed introduction to Tivoli Distributed Monitoring 3.6, although the examples in this redbook were performed with Tivoli Distributed Monitoring 3.6. We used, however, an early version of that product and it can therefore not be guaranteed that the generally available (GA) version of Tivoli Distributed Monitoring is 100% identical with the version we work with in this redbook. Wherever necessary, we point out where to be careful. All the monitoring examples we explain, do also apply to previous versions of Tivoli Distributed Monitoring.

We assume that the reader is already somewhat familiar with the basic principles of TME 10 and Tivoli Distributed Monitoring.

More information regarding these topics and more introductory material can be found in the following redbooks:

- *TME 10 Cookbook for AIX - Systems Management and Networking Applications*, SG24-4867
- *A First Look at TME 10 Distributed Monitoring 3.5*, SG24-2112
- *Migrating from Systems Monitor for AIX to TME 10 Distributed Monitoring*, SG24-4936

To read more about the Lightweight Client Framework and LCF-enabled applications you can also refer to the following redbooks:

- *Tivoli Software Distribution 3.6: Unleashing the Power of TME and LCF*, SG24-2045 (Available at a later date.)
- *TME 10 Framework Version 3.2: An Introduction to the Lightweight Client Framework*, SG24-2025

1.1 What is TME 10?

We only give a brief overview of Tivoli Management Environment (TME) 10 in this section. For a detailed explanation of Tivoli and its architecture, refer to the redbook *Understanding Tivoli's TME 3.0 and TME 10*, SG24-4948.

TME 10 is the resulting product of the merger between IBM's systems management division and Tivoli Systems, which formed Tivoli Systems, an IBM company. The product set is therefore the result of combining Tivoli's TME 3.0 with IBM's SystemView family of products.

TME 10 is based on the Tivoli Management Framework (TMF) that provides common services to Tivoli applications. On top of this framework reside a number of Tivoli core applications that utilize services provided by the TME 10 Framework. These applications themselves provide essential systems management functions that are quite generic and create the basis for management of any system, network, application or database.

These core applications are:

- Tivoli Software Distribution
- Tivoli Distributed Monitoring
- Tivoli User Administration
- Tivoli Security
- Tivoli Enterprise Console
- Tivoli Inventory

Systems Management and therefore the core applications can be divided into management disciplines. The disciplines defined by Tivoli are as follows:

- Deployment
- Availability
- Security
- Operations

1.2 What is Tivoli Distributed Monitoring?

Tivoli Distributed Monitoring belongs in the Availability area, together with Tivoli Enterprise Console. In the IBM SystemView family of products there was also a monitoring application that was called Systems Monitor and was based on SNMP monitoring.

Systems Monitor consisted of a monitoring component for systems, called System Information Agent (SIA) and a Mid-Level Manager (MLM) component for SNMP. While MLM is now part of Tivoli NetView, SIA is no longer available, as its function can be completely covered by Tivoli Distributed Monitoring. To learn more about this topic you can refer to the redbook *Migrating from Systems Monitor for AIX to Tivoli Distributed Monitoring*, SG24-4936.

Tivoli Distributed Monitoring is based on the Tivoli core application that was formerly known as Tivoli/Sentry. It is based on the Tivoli Framework and continues to be the Tivoli core application for monitoring systems, while Tivoli NetView and MLM continue to be the Tivoli application for monitoring networks. Both products are integrated with Tivoli Enterprise Console as the central point of control.

1.3 Tivoli Distributed Monitoring 3.6

In this book we work with Tivoli Distributed Monitoring 3.6, however, the examples of creating custom monitors can be applied to Tivoli/Sentry 3.0.2 also.

The main new feature in Tivoli Distributed Monitoring 3.6 is support for the Lightweight Client Framework concept. This support is provided for UNIX and Windows NT as LCF gateways in respect to Tivoli Distributed Monitoring and for UNIX, Windows NT and OS/2 as LCF endpoints.

Also provided is a new monitoring collection, specifically for OS/2.

Important Note

The examples in this redbook that deal with Version 3.6 of Tivoli Distributed Monitoring have been performed with an early version of that product. Therefore, it cannot be guaranteed that information regarding Tivoli Distributed Monitoring 3.6 is always fully compliant with the generally available (GA) version of that product. Where appropriate, we point out where to be careful about supported product functions.

1.4 Custom Monitors for Tivoli Distributed Monitoring

In this section we look briefly at why you might want to create custom monitors for Tivoli Distributed Monitoring and how this can be achieved.

1.4.1 Why Create Custom Monitors?

Tivoli Distributed Monitoring provides a wide range of monitors that are delivered with the product, such as monitors for operating system parameters, network parameters and application parameters. The monitoring collections that come with Tivoli Distributed Monitoring are also constantly enhanced; for example, Tivoli Distributed Monitoring 3.5 introduced new collections to monitor Sybase and Oracle RDBMS servers.

In a number of cases, however, it might be required to either extend the monitoring collections that come with Tivoli Distributed Monitoring or to create a completely new monitoring collection.

Some examples are:

- You want to monitor a device for which no monitor or monitoring collection exists.
- You want to monitor how a system or application meets Service Level Agreements (SLAs).
- You want to monitor special operating system or network characteristics that are not covered by the standard monitors shipped with Tivoli Distributed Monitoring.
- You want to monitor an operating system platform that is not directly supported by Tivoli Distributed Monitoring.

1.4.2 How to Create Custom Monitors

Creating a custom monitor can basically be divided into the following steps:

- Implementing the monitoring code
- Integrating the monitoring code into Tivoli Distributed Monitoring

1.4.2.1 Implementing the Monitoring Code

Implementing the monitoring code refers to writing a script or program that actually monitors a resource and makes a result available to the monitoring application. This result is usually a numeric value or a string.

The monitoring code that you implement can in principle be written in any programming language that is supported on the operating system platform that you are writing the monitor for and that can be translated into executable form.

For example, you can implement the monitor in Perl, Bourne shell or C. In fact, most of the monitoring collections that are shipped with the Tivoli Distributed Monitoring product are implemented in Perl.

Note

Even if you want to write a monitor for Windows NT you can use Perl, as a Perl interpreter is part of the TME 10 Framework.

If you are implementing a Tivoli Distributed Monitoring proxy, you have to use a programming language that is supported on the managed node that acts as the proxy. For example, if you are monitoring a router for which the proxy resides on AIX, you can use any programming language that is supported on AIX.

In the case of writing a monitor for a router, you can also use the commands for SNMP management that are shipped with Tivoli Distributed Monitoring, such as wsnmpget and wsnmpset.

The language you choose will depend on factors such as ease-of-use, performance, security, and so on.

1.4.2.2 Integrating the Monitoring Code into Tivoli Distributed Monitoring

Once you have completed writing custom monitoring code you will need to integrate the code into Tivoli Distributed Monitoring, that is make the code accessible to Tivoli Distributed Monitoring.

The easiest way to do that is to use the Numeric script and String script monitors from the Universal monitoring collection.

Figure 1. Integrating a Custom Monitor

In the above figure you see the window you get when you add a monitor to a Tivoli Distributed Monitoring profile. In the Monitoring Collections section we have selected **Universal**, which lists all the monitors in the Universal monitoring collection in the Monitoring Sources section.

The two monitors of interest are Numeric script and String script. These two monitors are basically dummy monitors or stubs for actual monitoring code. You can provide the code for these monitors yourself and then specify the program or script that implements the monitoring code in the Program field in the Monitor Arguments section.

In the example we have clicked on **Numeric script**, which displays the parameters for that monitor in the Monitor Arguments section. We have entered `/home/stefan/custom_monitor.sh` in the Program field, indicating that the implementation for our new monitor resides in this file.

Our monitor in this case is implemented in Bourne shell, indicated by the file type .sh, but it could also be implemented, for example, in C. The only requirement for the implementation of a Numeric script monitor is that the monitor returns a numeric value. For a String script monitor, respectively, the return value needs to be a string.

Note

If you implement your monitor in a language that needs to be compiled, (for example, C) then before using the monitor in the Numeric script or string script monitors you of course need to compile the monitoring code (in this case a .c file) to create executable code first. This step is not necessary for a shell script or Perl script, as these are interpreter languages.

Although the generic script monitors provide the easiest way to include your own custom monitors in Tivoli Distributed Monitoring, they are somewhat limited when you want to create more sophisticated monitors.

Some of the limitations are:

- You cannot provide run time parameters to the monitoring code in a convenient way. Of course, you can enter command line options when specifying the monitor command in the Program field. However, you cannot provide, for example, a GUI to enter values or offer alternatives. When looking at the monitors that are shipped with Tivoli Distributed Monitoring you will often find that the monitor offers to enter parameters in the GUI, such as the following:

Figure 2. Parameters for a Monitor

In the above example, we specified the daemon to monitor (oserv) in the Daemon field. You see that although Program is an argument to the Numeric script monitor, the new monitor we implement with custom_monitor.sh cannot receive parameters like this.

- You cannot create new monitoring collections.
- You need to take care of distributing the monitoring code (in our case custom_monitor.sh) to the node where it is to be executed.
- You cannot install the new monitor conveniently in another TMR.

The above issues are addressed by Monitoring Capability Specification Language (MCSL).

MCSL allows you to do the following:

- Create new monitoring collections and monitors.
- Furnish monitors with a GUI to enter parameters.
- Automatic deployment of monitoring code.
- Easy installation of monitoring collections in other TMRs.

For using MCSL you need nothing besides Tivoli Distributed Monitoring 3.6 and the Tivoli Application Development Environment (ADE), which is part of the Tivoli Framework.

With MCSL you still write your actual monitoring code in your favorite language, such as Perl, Bourne shell or C. MCSL is then used to wrap your monitoring code and make it available to Tivoli Distributed Monitoring. This is done by using the mcsl command, which is part of Tivoli Distributed Monitoring.

This command is used to translate MCSL code into a monitoring collection and also to install the monitoring collection in a TMR. We will show more detailed examples later in this book but to give you a first impression the following figure shows an example written in MCSL language:

```
#include "Sentry2_0.dsl"
#include "cpu.dsl"
Collection "UnixCPU" {
    CodeID = "$Id:cpu.csl,v 1.0$";
    Version = "1.0";
    Require = ">2.0.2";
    HelpMessage = (cpu_generic_help);
    EventBaseClass = "Sample_Sentry_Monitors";
    NoticeGroup = "Sentry";
#include "operators.csl"
#include "choicelists.csl"
#include "formats.csl"
    ChoiceList DiffOptions {
      ButtonLabel = (cpu_ButtonLabelChoice);
      {
         { (cpu_cpu_choice_idle) "-idle" }
         { (cpu_cpu_choice_user) "-user" }
         { (cpu_cpu_choice_system) "-system" }
         { (cpu_cpu_choice_system_user) "-system_user" }
      };
    };
    Monitor cpuload Numeric Group numeric {
      Description = (cpu_cpuload_Descr);
      ValueDescription = (cpu_cpuload_val_Descr);
      HelpMessage = (cpu_Busy_Help);
      Argument (cpu_cpu_list)
        RestrictedChoice "DiffOptions"
        DefaultValue "-idle";
      Implementation (aix3-r2, aix4-r1)
      Shell("/bin/sh", "-c", Command, "CPU_LOAD")
      Import "cpuload.sh"
     ;
      Implementation (hpux9, hpux10)
      Shell("/bin/sh", "-c", Command, "CPU_LOAD")
      Import "cpuload.sh.hp"
     ;
      Implementation (sunos4, solaris)
      Shell("/bin/sh", "-c", Command, "CPU_LOAD")
      Import "cpuload.sh.sun"
     ;
   };
}
```

Figure 3. Example MCSL Source File

You can see in the highlighted rows in the above figure that the actual monitor implementations are still shell scripts that are wrapped in the MCSL code.

In this example you see another advantage of MCSL: you can provide different implementations for the same monitor, depending on the operating system platforms (AIX, HP-UX and Solaris in the above example).

The rest of the above source code defines, for example, the input fields that the monitor offers to enter parameters.

1.4.3 Techniques for Monitoring Devices that are not Directly Supported

You might need to monitor devices or operating systems that are not directly supported by Tivoli Distributed Monitoring, that is you either need to monitor an operating system that can either act as an LCF endpoint or managed node, or you want to monitor, for example, a hardware device that does not provide a full blown operating system at all.

We show several techniques of how to deal with these issues in this redbook. For example, we show in 3.10, “Monitoring an IBM 8235” on page 99 how to use the UNIX Expect tool in combination with the telnet feature of an IBM 8235 hardware device that allows us to monitor parameters of that device even though it does not have an operating system supported by Tivoli Distributed Monitoring.

We show a similar approach in 3.11, “Monitoring Performance of a VM System” on page 111 where we show a custom monitor for VM, an IBM mainframe operating system that is not supported by Tivoli at this time.

1.5 How to Read This Book

You can either read this book from cover to cover. or refer to any section of interest. We start in Chapter 2, “Setting up the Monitoring Environment” on page 11 with setting up our Tivoli environment in which we perform the example scenarios in this book.

We explain the differences between Tivoli Distributed Monitoring 3.6 and previous versions and perform a simple monitoring example to verify that our environment works as expected. If you are already familiar with the basic operation of Tivoli Distributed Monitoring 3.6, you can skip this chapter.

In Chapter 3, “Custom Monitoring Examples” on page 37 we present all the custom monitors we created during this project. You can refer to any specific monitor of interest. However, if you are new to creating monitors for Tivoli Distributed Monitoring we recommend that you read this chapter from the beginning as we develop some techniques that are used throughout the chapter.

In Chapter 4, “Creating a Tivoli Installable Image for our Custom Monitors” on page 191 we show how to create a Tivoli installable image that contains all the monitoring collections that we develop in this redbook. This image can then be installed, as any other Tivoli software, from the Tivoli desktop.

In Chapter 5, “Troubleshooting” on page 213 we give some hints that help in problem determination when creating custom monitors for Tivoli Distributed Monitoring.

1.6 Getting the Code Examples in this Book

To obtain the source code for all the example monitors described in this redbook, you can either point your Web browser to www.redbooks.ibm.com or send an e-mail to uelpenich@us.ibm.com.

Chapter 2. Setting up the Monitoring Environment

In this chapter, we describe the setup and configuration that we use in creating monitors for Tivoli Distributed Monitoring 3.6. We use this environment in subsequent chapters.

The assumption made in this chapter is that the user is familiar with the concepts of the Tivoli Framework and the basic principles of Tivoli Distributed Monitoring. If you are familiar with Tivoli Distributed Monitoring 3.6 you can skip this chapter.

2.1 System Overview and Objectives

To accomplish the objective of creating custom monitors using Tivoli Distributed Monitoring 3.6 our setup consists of a combination of UNIX, Windows NT and Windows 95 machines. An explanation of the setup is described following the diagram:

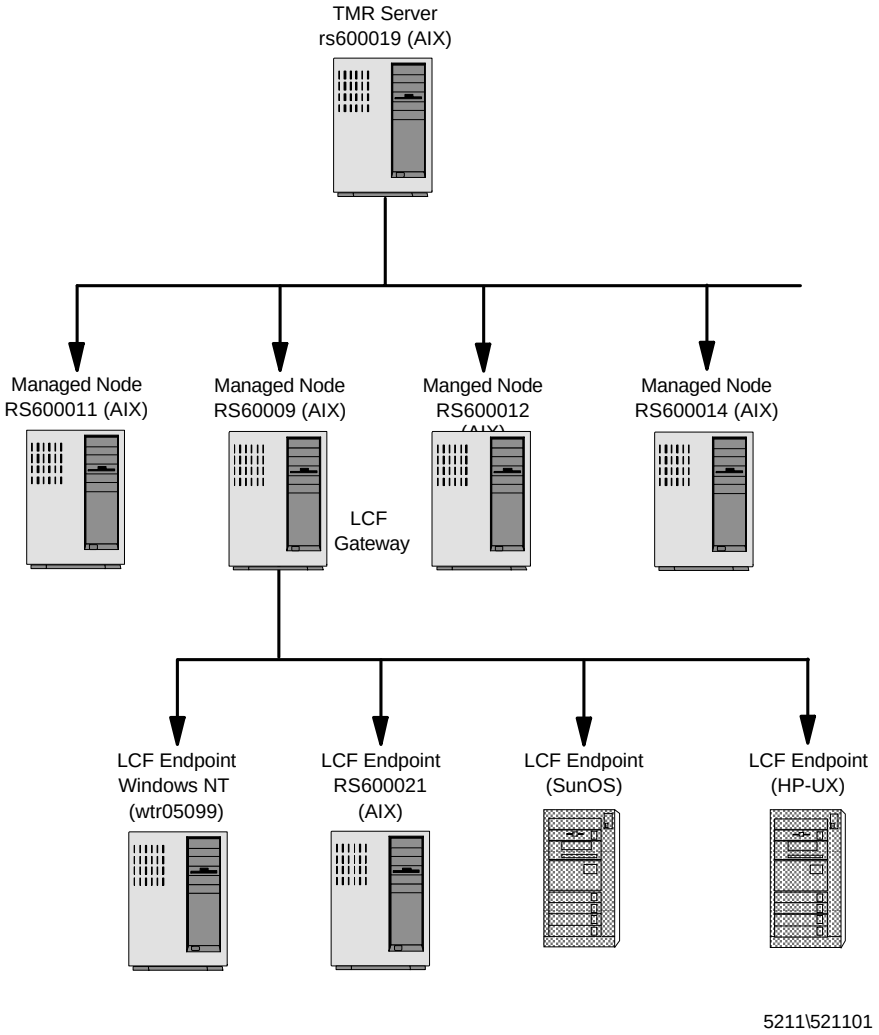


Figure 4. Lab Environment

The AIX host rs600019 is set up as a TMR server by installing TME 10 Framework 3.2 on it. We upgrade the Tivoli Framework on this host to TME 10 Framework 3.2.1.

Note

At the time this redbook is written, we were working with an early version of the Tivoli Framework that supports LCF-enabled applications, which in our environment is called TME 10 Framework 3.2.1. In the generally available version this product is called Tivoli Framework 3.6.

We then create four managed nodes on this host and make one of the managed nodes an LCF gateway (rs60009). We then attach another UNIX host (rs600021) and an NT host (wtr05099) as LCF endpoints to the LCF gateway.

Our second configuration setup is done in the following manner. A Windows NT server (foxtrot) is set up as a TMR server. We create two NT hosts (victor and juliet) as managed nodes from this TMR server (foxtrot). We configure victor also as an LCF gateway that connects to the TMR server (foxtrot). We create an LCF endpoint which is a Windows 95 host and attach it to the LCF gateway (victor). The following diagram illustrates our second environment.

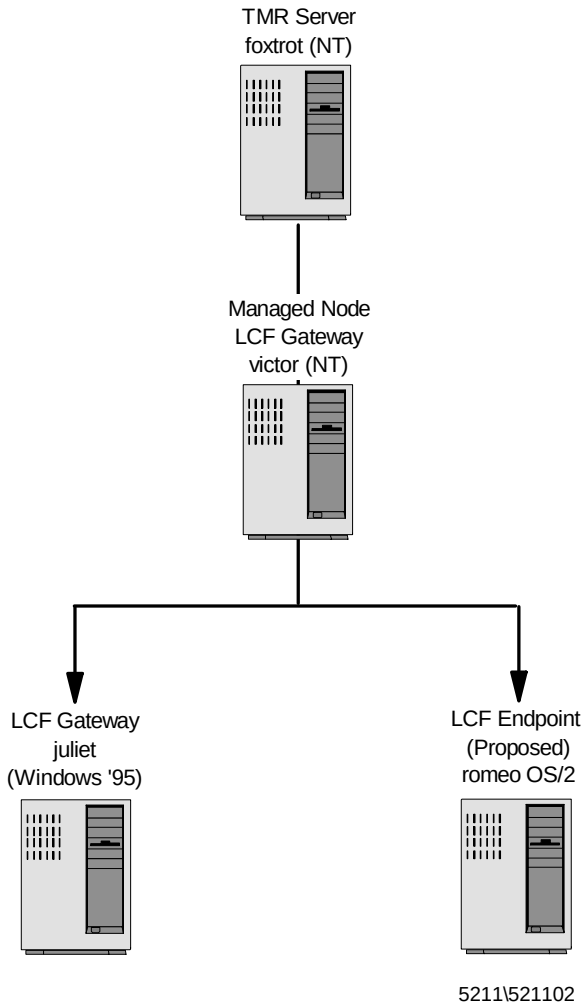


Figure 5. Lab Environment

2.1.1 Updating to TME 10 Framework 3.6 on AIX

To work with Tivoli Distributed Monitoring 3.6, it is necessary to install the Tivoli Framework. In our case we install the TME 10 Framework 3.2 and then perform an upgrade to TME Framework 3.2.1 by installing a patch.

> **Note**
>
> At the time of this writing, the TME Framework was still at a release level of 3.2 but it is likely that when the code is generally available it will be called TME Framework 3.6.

The next sequence of steps shows how we install the TME 10 Framework 3.6 environment on an AIX 4.2.0 system. We assume that the user is using TME Framework 3.2 and is updating to a TME Framework 3.6 environment.

1. Run the Tivoli environment script:

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

2. Back up the object database. This is not mandatory but is highly recommended. By default, the backup is stored in the $DBDIR/backups directory.

   ```
   # wbkupdb
   ```

3. Start the Tivoli desktop from the command line. If you are not starting from the console, make sure your DISPLAY variable is set to your host IP address.

   ```
   # tivoli
   ```

4. In the Tivoli Desktop select **Desktop** from the menu bar and then select **Install->Install Patch...** from the pull-down menu. The following window will appear:

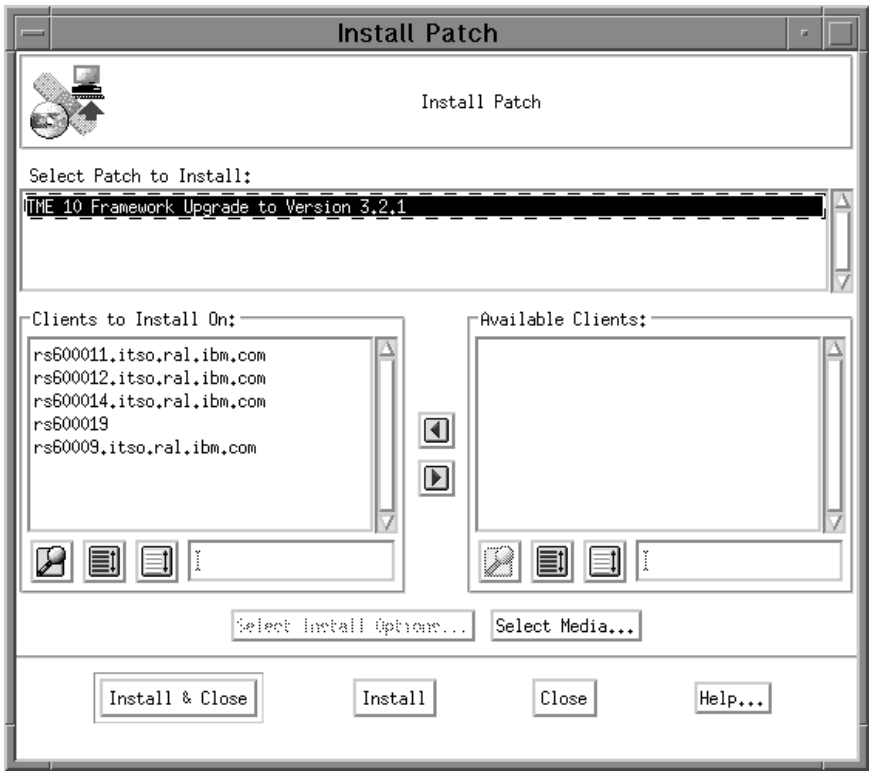


Figure 6. Upgrading to TME 10 Framework 3.6

Note

In the generally available product this patch will be called Tivoli Framework Upgrade to Version 3.6.

5. Click on the **Select Media** option and set the media to where the TME 10 Framework 3.2.1 patch is located.
6. From the window select the patch you want to install. This should be **TME 10 Framework Upgrade to Version 3.2.1**.
7. From the list of Available Clients select the clients for installation and move them to the Clients to Install On: listbox.
8. Select the **Install & Close** option and continue with the installation after verifying that the installation procedure has found no problems with the clients that the patch is being installed on.

2.1.2 Installing TME 10 Distributed Monitoring 3.6 on AIX

After the installation for TME 10 Framework 3.2.1 is complete follow the sequence of steps outlined to install TME 10 Distributed Monitoring 3.6 on the same clients listed in the window in Figure 6.

1. In the Tivoli Desktop select **Desktop** from the menu bar and then select **Install->Install Product...** from the pull-down menu.

   You will see the following window:

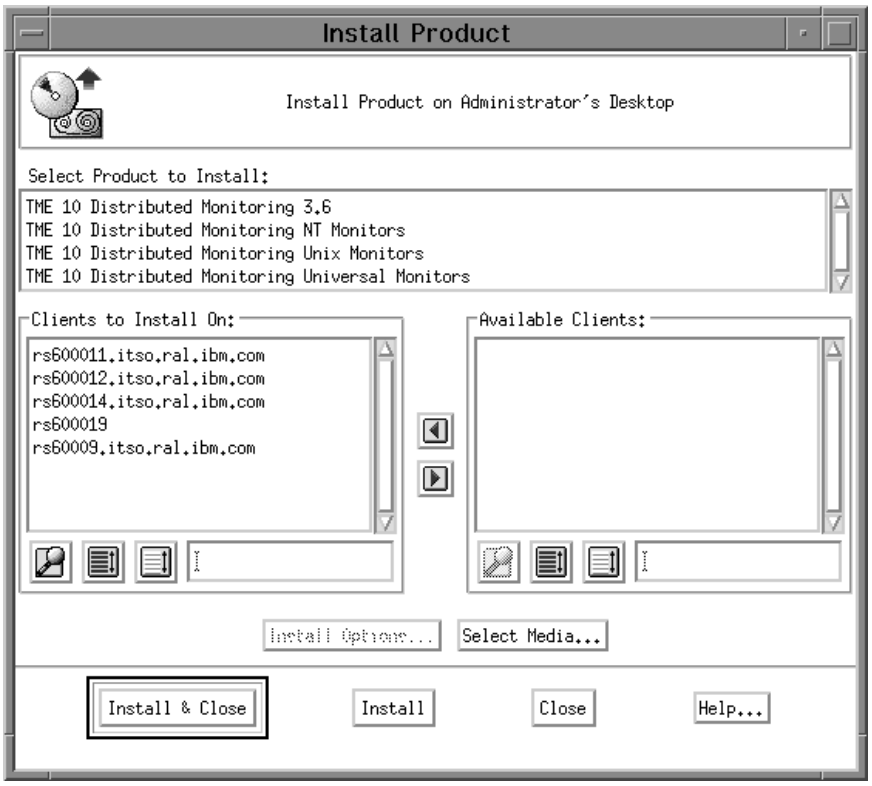


Figure 7. Installing TME 10 Distributed Monitoring 3.6

2. Select **TME 10 Distributed Monitoring 3.6** from the list displayed and select **Install & Close** from the dialog window. This product needs to be installed on all managed nodes that are to be monitored by Tivoli Distributed Monitoring or that act as an LCF gateway, providing Tivoli Distributed Monitoring function to LCF endpoints. In our case, we install Tivoli Distributed Monitoring 3.6 on all our managed nodes.
3. Click on **Set** in the next window to set the default path to receive the Tivoli header files. These header files are now part of the Tivoli Distributed Monitoring and are used by Monitoring Capability Specification Language. By default, the files are installed in /usr/local/Tivoli/include.

Figure 8. Install Options Window

Note

The header files Sentry2_0.dsl, choicelists.csl, formats.csl and operators.csl are used in writing Monitoring Capability Specification Language scripts.

4. The system verifies the condition of the clients about to receive the TME 10 Distributed Monitoring 3.6 software. Verify that there are no errors and select the **Continue Install** option from the window.

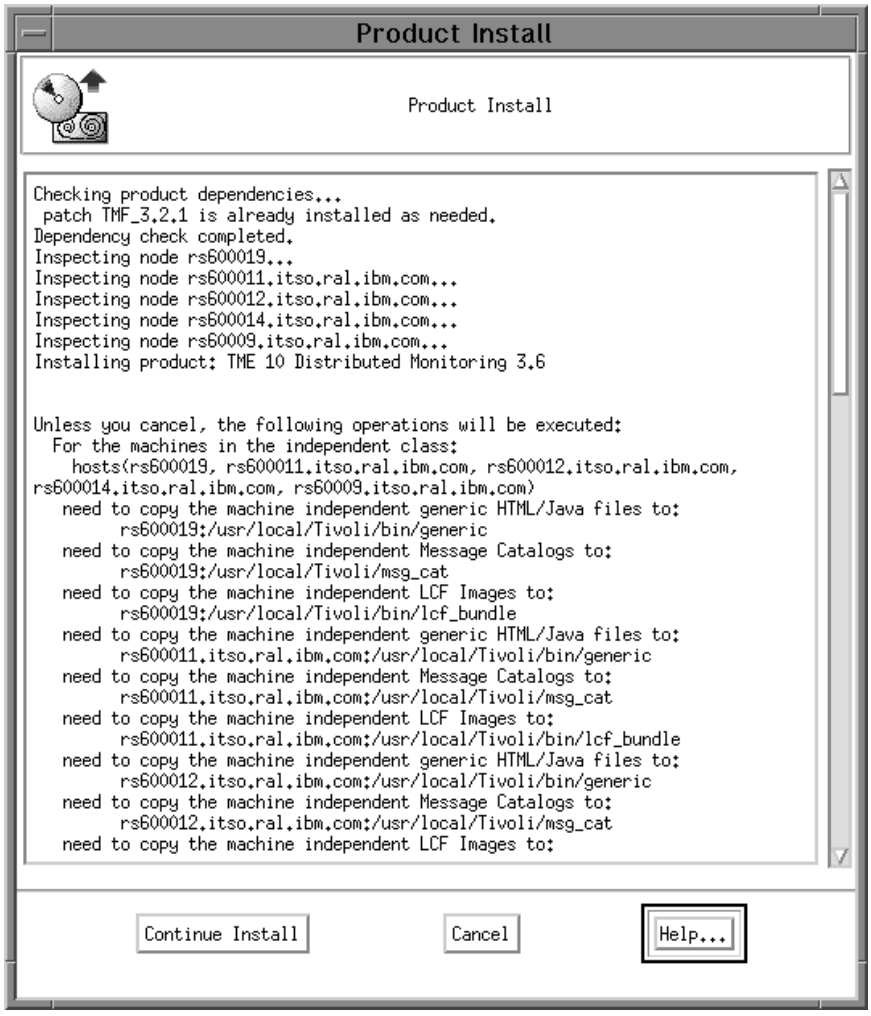


Figure 9. Installing TME 10 Distributed Monitoring 3.6

5. In order to verify that the monitoring software is performing as specified we install the Universal Monitors on the TMR server. The steps for the installation are the same as for the Tivoli Distributed Monitoring 3.6 application.

Finally, we type the following command from the command line:

```
# wlsinst -a
```

The output from this command shows that we did install the products successfully on the host rs600019. The output from the above command is shown in the following figure.

```
*-------------------------------------------------------------------------------*


TME 10 Framework Upgrade to Version 3.2
TME 10 Framework Upgrade to Version 3.2.1
root@rs600019:/usr/local/Tivoli/include/aix4-r1/tivoli# wlsinst -a
*-------------------------------------------------------------------------------*

                                Product List
*-------------------------------------------------------------------------------*


TME 10 Framework
TME 10 Distributed Monitoring 3.6
TME 10 Distributed Monitoring Universal Monitors

*-------------------------------------------------------------------------------*

                                 Patch List
*-------------------------------------------------------------------------------*


TME 10 Framework Upgrade to Version 3.2
TME 10 Framework Upgrade to Version 3.2.1
root@rs600019:/usr/local/Tivoli
```

Figure 10. Output from wlsinst -a Command

2.1.3 Installing TME 10 Framework on Windows NT

The following sequence of steps outline what we did to set up our Windows NT environment:

1. Click on **Start->Run** from the Windows NT Desktop. Enter:

   ```
   d:\setup.exe
   ```

   where d: is the drive letter for the TME 10 Framework 3.2.1 CD-ROM drive. Select **Next>** from the window displayed.

2. Enter the Name and Company information in the next window. Select **Next>** to continue.

3. Enter your installation password in the next window and select **Next>** to continue.

4. Enter your user name and password for remote file access in the next window. You can leave this field blank and just click on **Next>** if your system is not going to access any remote files.

5. Click on **Custom** in the next window display. Click on **Next>** to continue.

6. Select each item from the next window for installation. Click on **Next>** to continue.

7. Enter your license key in the next window. Click on **Next>** to continue.

8. Click on **Next>** in the window to accept the default path for the Tivoli database directory.

9. Verify from the next window that the `Server database installation completed successfully` message is displayed.
10. Click on **OK** in the next window to confirm that the installation is completed.

> **Note**
>
> After successfully installing TME 10 Framework 3.2 we then upgraded the NT host by installing the TME 10 Framework 3.2.1 patch as we described in the steps for the AIX platform. This is a prerequisite for working with TME 10 Distributed Monitoring 3.6.

2.1.4 Installing TME 10 Distributed Monitoring 3.6 on Windows NT

The following sequence of steps outline the process for installation of TME 10 Distributed Monitoring 3.6 on the NT platform:

1. In the Tivoli Desktop select **Desktop** from the menu bar and then select **Install->Install Product...** from the pull-down menu.

   You will see the following window:

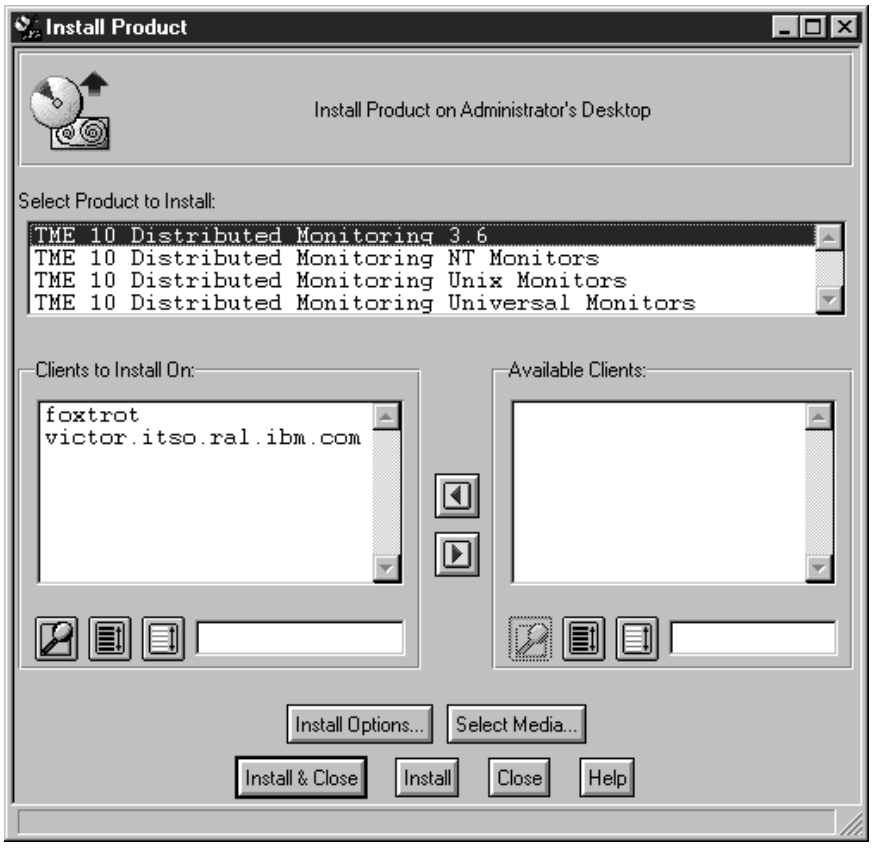


Figure 11. Installing TME 10 Distributed Monitoring 3.6 on Windows NT

2. Select **TME 10 Distributed Monitoring 3.6** from the list displayed and select **Install & Close** from the dialog window.
3. Click on **Set** to set the default path for the Tivoli include files.
4. Click on **Continue Install** after verifying that there are no problems found on the client nodes that the product is going to be installed on.

5. Verify that the TME 10 Distributed Monitoring 3.6 installation was successful.
6. Repeat step 1 and install Tivoli Distributed Monitoring NT Monitors on the TMR server.

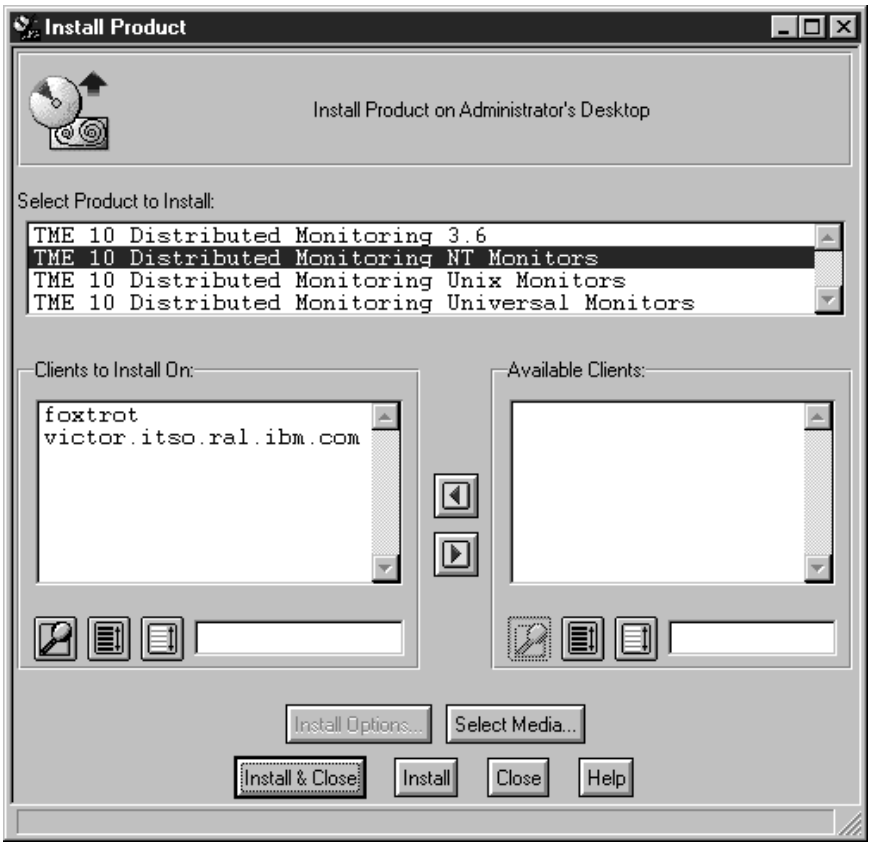


Figure 12. Installing Tivoli Distributed Monitoring NT Monitors

7. Click on **Tivoli Distributed Monitoring NT Monitors** in the listbox. The monitoring collection needs to be installed only on the TMR server.
8. Click on **Install & Close** to continue with the installation of the NT monitors.
9. Click on **Continue Install** after verifying that there are no problems found on the client nodes that will be receiving the distribution.

2.2 Setting Up the LCF Environment

The TME 10 Framework 3.2.1 (or 3.6 respectively) is packaged with the following components that support the LCF environment. These components are:

- LCF endpoint manager
- LCF endpoint
- LCF gateway

The following steps show the process of creating an LCF endpoint using both the GUI and the command line interface.:

2.2.1 Creating an LCF Gateway on Windows NT

After the installation of a TMR server (foxtrot) and a managed node (victor) in the NT environment we proceed to set up a managed node (victor) also as an LCF gateway. The sequence of steps describe how we create the LCF gateway using the Tivoli GUI.

1. In the Tivoli Desktop, select the **EndpointManager** icon and click the right mouse button. The following menu is displayed:

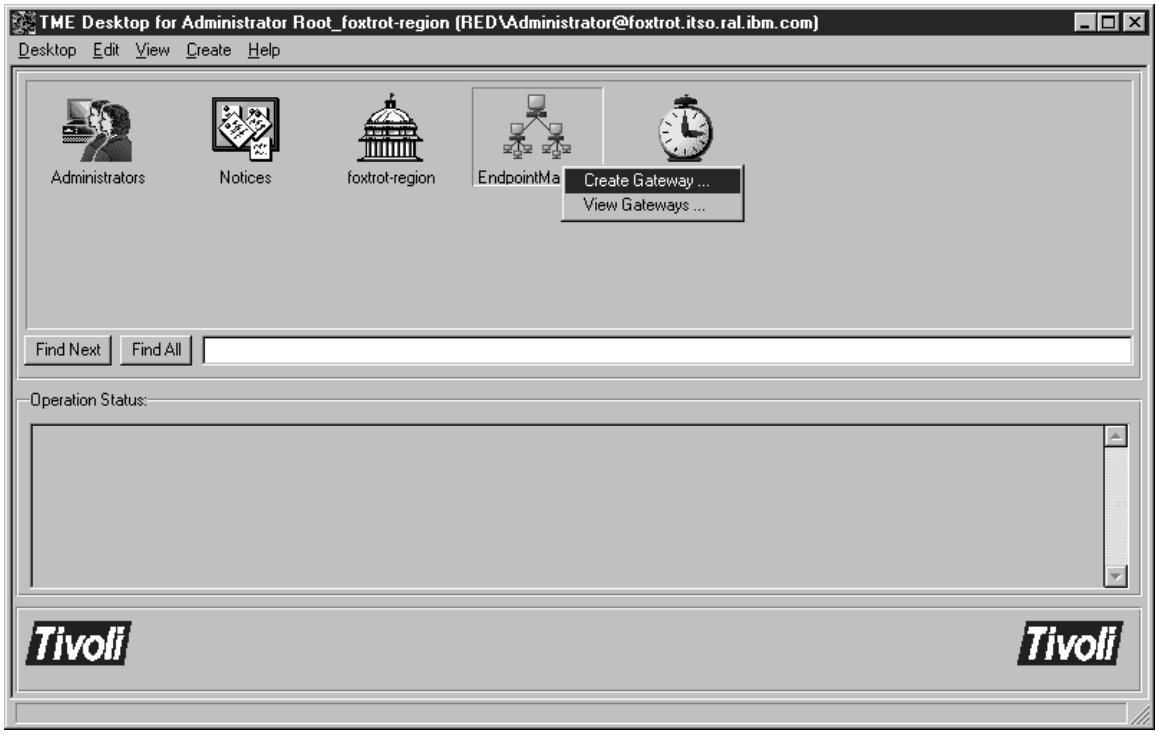


Figure 13. Tivoli Desktop

2. Click on the **Create Gateway...** menu option. In the Create Gateway dialog, enter the name of the gateway, which is a matter of your own choice. The default port number is 9494. Enter the name of the managed node in the Managed Node Proxy listbox. You can either enter the node name or select it from the list after clicking on **Managed Nodes** and selecting it from the choices displayed.

Figure 14. Creating an LCF Gateway on Windows NT

Select **Create & Close** from this dialog.

3. The LCF gateway will be created on victor.itso.ral.ibm.com.
4. After the LCF gateway creation, double-click on the **EndpointManager** icon on the Tivoli desktop to list the LCF endpoint that connects to the LCF gateway.

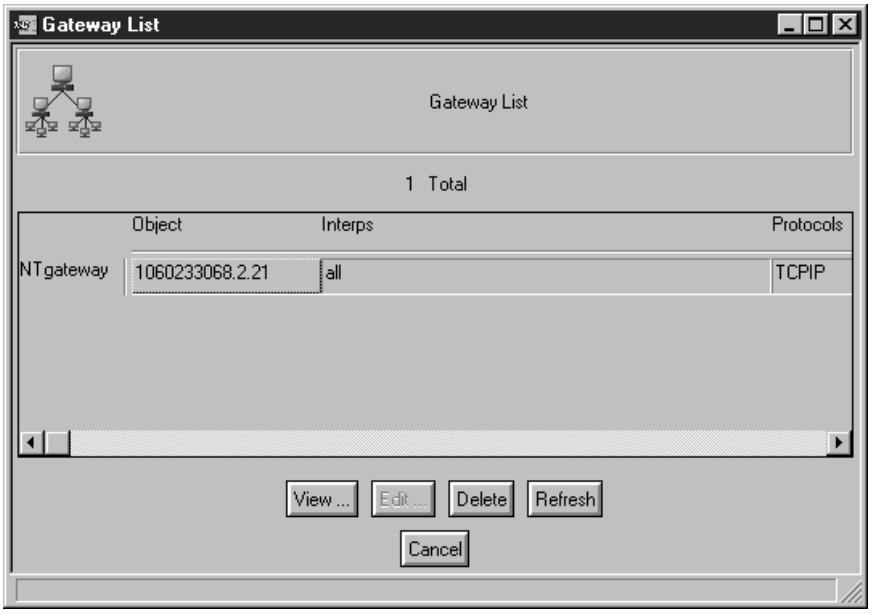


Figure 15. Gateway List Window

5. If there were any LCF endpoints connected to the LCF gateway selecting **View** would list the LCF endpoints.

We use the same process described above to create an LCF gateway (rs60009) in the AIX environment as shown in Figure 4 on page 11.

2.2.2 Creating an LCF Gateway from the Command Line

The following steps show how to install an LCF gateway for AIX from the command line:

1. Run the tivoli environment setup script:

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

2. Back up the object database. This is not mandatory but recommended. By default, the backup is stored in the $DBDIR/backups directory:

   ```
   # wbkupdb
   ```

3. Use the command `wcrtgate` to create the LCF gateway:

   ```
   wcrtgate -h rs60009 -n AIX-LCF-Gateway
   ```

   The -h parameter specifies the managed node and the -n parameter specifies the name of the gateway. This command will create a new object in the database:

   ```
   1056670195.2.25#TMF_Gateway::Gateway
   ```

4. Verify that the LCF gateway is created successfully using the command `wgateway`:

   ```
   wgateway AIX-LCF-Gateway describe
   Object      : 1056670195.2.27#TMF_Gateway::Gateway#
   Hostname    : rs60009
   Port        : 9494
   Timeout     : 300
   ```

5. Since the creation of the LCF gateway modifies the object database we recommend backing up the database again:

   ```
   wbkupdb
   ```

2.2.3 Installing the LCF Endpoints

As seen from Figure 4 on page 11 a combination of UNIX, NT, OS/2 and Windows 95 machines were selected to be the LCF endpoints.

2.2.3.1 Installing the LCF Endpoint on UNIX

1. Installation of an LCF endpoint in UNIX can only be done from the command line. The following command installs an LCF endpoint on a UNIX host:

   ```
   winstlcf -g 9.24.104.249+9494 aix
   ```

   The -g option forces the LCF endpoint to log in to the LCF gateway specified by the IP address following this parameter. The LCF gateway must also be a managed node.

2. To verify that the LCF endpoint got connected to the LCF gateway, click on the **EndpointManager** icon on the Tivoli desktop.

3. From the list of gateways select the gateway whose IP address is listed above and click on **View...**.

4. The list that is displayed shows the LCF endpoints of the LCF gateway that was selected for viewing. Verify that your LCF endpoint created above is in the list.

2.2.3.2 Installing the LCF Endpoint on Windows NT

An LCF endpoint for NT can be either set up locally using the TME 10 Framework CD-ROM and running the setup.exe from the directory /PC/lcf/winnt on the CD-ROM, or you can also FTP the files to your local NT machine and run setup.exe from there. The setup.exe uses InstallShield to install the LCF endpoint code on your NT machine. The following sequence of steps outlines the process:

1. Run `setup.exe` from the D:\PC\LCF\WINNT directory. The D: drive in this case should reference your CD-ROM drive that has the TME 10 Framework.
2. Figure 16 shows the opening window of the installation for the LCF endpoint.

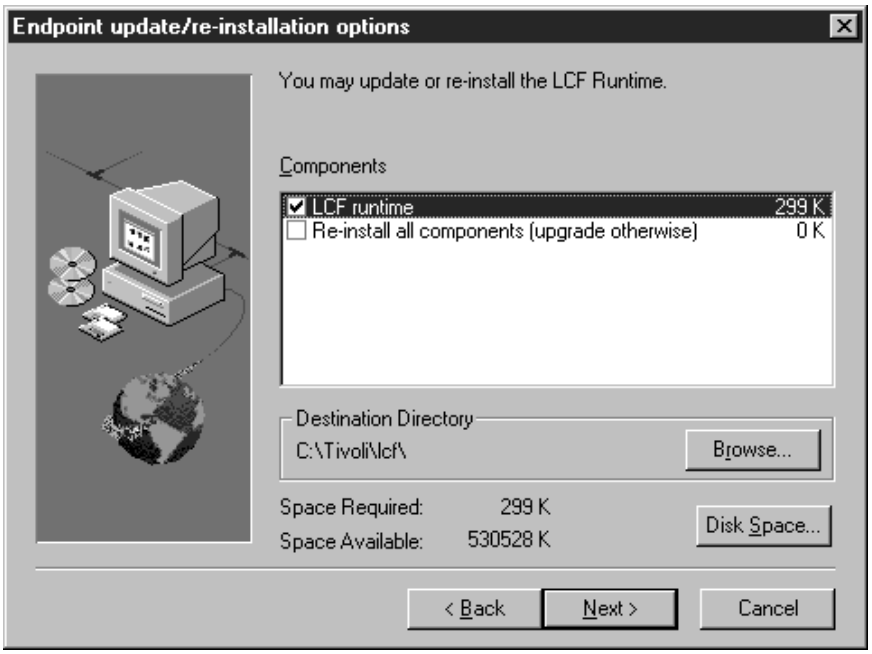


Figure 16. Installing LCF Endpoint Locally on Windows NT (1/5)

Click **Next>** to continue.

3. In the next window enter your user name and password for remotely accessing any files. Leave the fields blank if your system is not accessing any remote files. Click on **Next>** to continue. You will get the following dialog:

Figure 17. Installing LCF Endpoint Locally on Windows NT (2/5)

Enter the IP address of the gateway that the LCF endpoint will connect to in the Gateway listbox.

By default the port number that the endpoint will connect to is 9494.

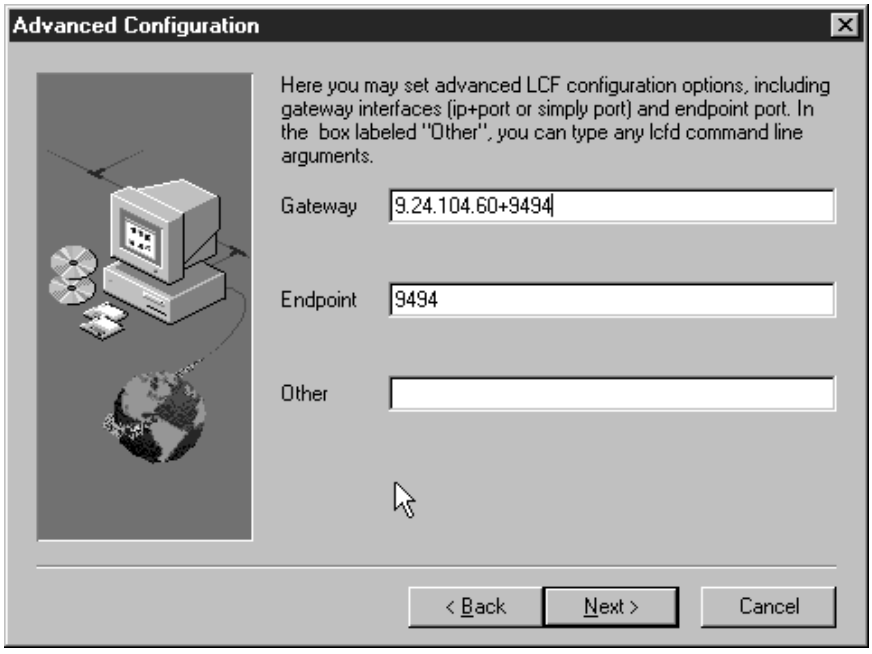


Figure 18. Installing LCF Endpoint Locally on Windows NT (3/5)

Select **Next>** to continue.

4. The next dialog shows that the LCF endpoint succeeded in logging to an LCF gateway.

Figure 19. Installing LCF Endpoint Locally on Windows NT (4/5)

Click on **Next>** to continue.

5. The next dialog informs you that the LCF endpoint install completed.

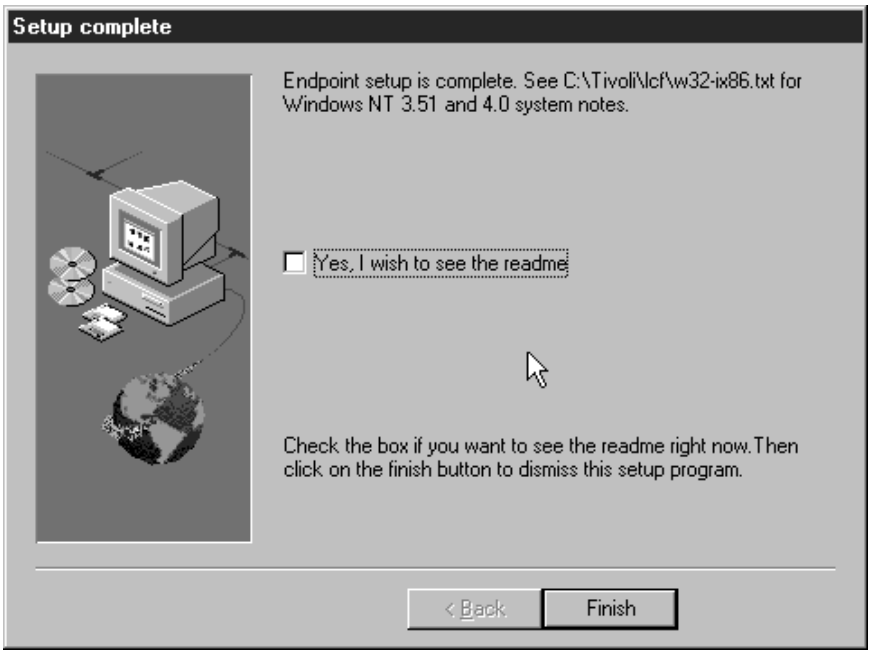


Figure 20. Installing LCF Endpoint Locally on Windows NT (5/5)

Click on **Finish** to complete the installation.

Note

The LCF endpoint installation on NT copies all the required files to the c:\Tivoli\lcf\bin\w32-ix86\mrt directory. The uninstall.bat file for the LCF endpoint resides in the c:\tivoli\lcf directory.

2.2.3.3 Installing the LCF Endpoint on Windows 95

Installing an LCF endpoint on Windows 95 is identical to installing it on a Windows NT host.

The setup.exe for installing an LCF endpoint for Windows 95 resides in the \PC\LCF\WIN95 directory of the CD-ROM for TME 10 Framework.

2.3 A Simple Monitoring Example

Having set up the environment as we describe in Figure 4 on page 11 we run a simple monitoring test to verify that our setup is performing correctly. However, since the Tivoli architecture has changed with respect to LCF endpoints, we need to create a dataless profile manager. We also show the data flow during a Tivoli Distributed Monitoring profile distribution to an LCF endpoint.

2.3.1 Creation of a Dataless Profile Manager

The concept of a dataless profile manager stems from the LCF architecture. Since the LCF endpoint has no object database, when the TME 10 Distributed Monitoring 3.6 profile gets distributed to an LCF endpoint it does not update any local object database as it does in the case of a managed node. Therefore, we create a profile manager for the LCF endpoint that is slightly different from a regular profile manager created for a managed node.

The difference in the process is shown below:

1. In the Tivoli desktop select the **rs600019-region** icon and double-click with your left mouse button to open this policy region.
2. In policy region window select **Create** from the menu bar and then select **Profile Manager...** from the pull-down menu.

   The following window will be displayed:

Figure 21. Creating a Dataless Profile Manager

3. Enter the name of the profile manager that you want to create and check the **Dataless Endpoint Mode** checkbox in the window. Select **Create & Close**.
4. The new profile manager (lcf) gets created and looks slightly different from the regular profile manager (classic).

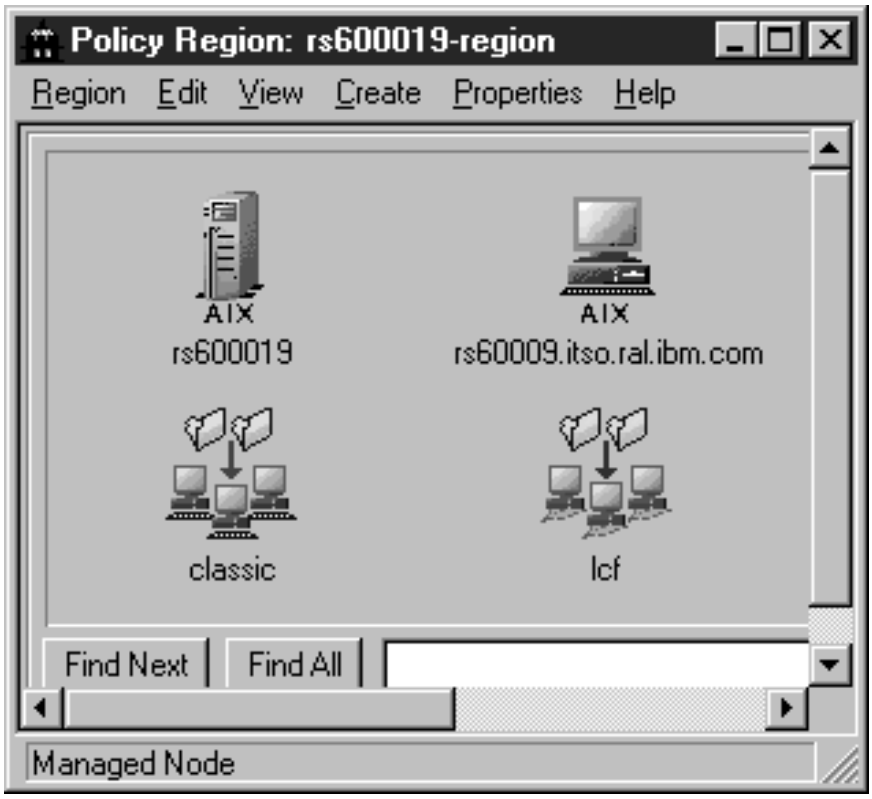


Figure 22. Policy Region Window

5. Double-click on the dataless profile manager you just created. Then select **Create** from the menu bar and then **Profile...** from the pull-down menu.
6. Give a name to the profile (lcfnt) and select **SentryProfile** as the type of the profile. Select **Create & Close**.
7. Select **ProfileManager** from the menu bar and then **Subscribers...** from the pull-down menu and add the Windows NT LCF endpoint (wtr0599) as a subscriber.

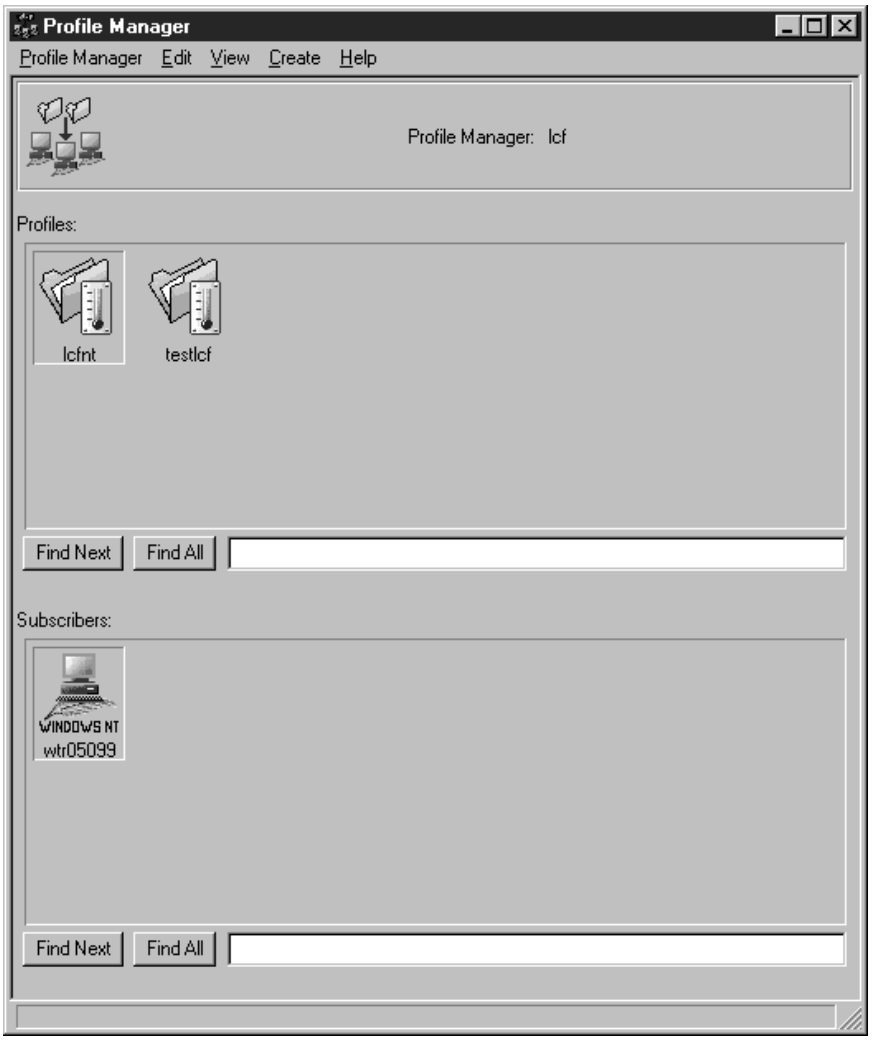


Figure 23. Profile Manager Window

8. Select the **lcfnt** icon from the window and double-click on it.
9. Select **Add Monitor** and add an **NT_LogicalDisk->Percent Free Space** monitor to it. Add `Always` as the Response level choice and check the **Pop Up** checkbox with the Administrator set to Root@rs600019-region. Set the monitoring schedule to 1 minute.
10. Select **Change & Close** and save the profile by selecting **Profile** from the menu bar and then **Save** from the pull-down menu.
11. Select **Profile** from the menu bar and then **Close** from the pull-down menu.
12. Back in the Profile Manager window select **Profile** from the menu bar and then **Distribute...** from the pull-down menu.

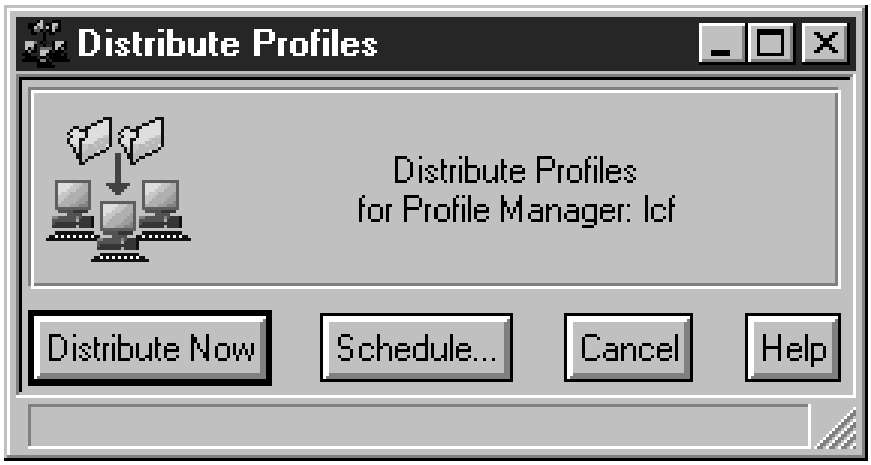


Figure 24. Distribute Profiles Window

Select **Distribute Now** from the window.

13. Wait for a minute or so and you should see the following dialog:

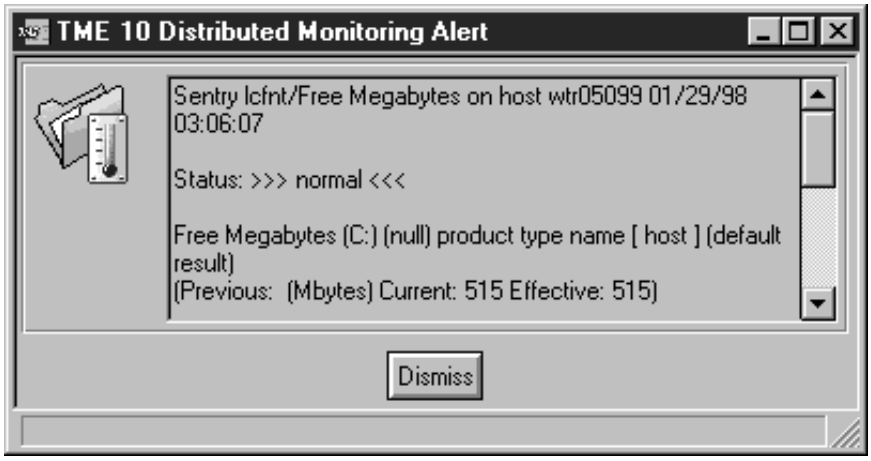


Figure 25. TME 10 Distributed Monitoring Alert Window

This shows that the LCF endpoint (wtr0599), a subscriber to the dataless profile manager, is responding to the TME 10 Distributed Monitoring 3.6 profile that was distributed to this endpoint. It shows that our LCF endpoint is set up and that the configuration is working for our simple monitoring example.

2.3.1.1 Key Files Distributed to an LCF Endpoint on Windows NT

Looking under the covers of an LCF endpoint on a Windows NT host reveals the *\Tivoli\lcf\dat* directory. An expansion of the directory tree under this branch reveals the following. The LCF endpoint has a node subdirectory that is listed below:

```
 C:\Tivoli\lcf\dat>dir

 Volume in drive C has no label.
 Volume Serial Number is 407E-0541

 Directory of C:\Tivoli\lcf\dat

02/03/98  01:43p        <DIR>          .
02/03/98  01:43p        <DIR>          ..
01/29/98  09:42a        <DIR>          01201822.138
02/03/98  01:43p                      0 dirlist
               4 File(s)              0 bytes
                           539,846,656 bytes free
```

The following directory structure is under the node subdirectory:

```
C:\Tivoli\lcf\dat\01201822.138>dir

 Volume in drive C has no label.
 Volume Serial Number is 407E-0541

 Directory of C:\Tivoli\lcf\dat\01201822.138

01/29/98  09:42a        <DIR>          .
01/29/98  09:42a        <DIR>          ..
01/29/98  03:08p                 1,018 .bk
01/20/98  06:37p        <DIR>          .sntcfg
01/20/98  06:37p        <DIR>          cache
01/29/98  03:08p                   350 DMInstlog
01/29/98  09:42a                   534 last.cfg
01/20/98  06:37p        <DIR>          LCF
01/29/98  09:42a                   440 lcf.dat
01/20/98  06:22p                    56 lcf.id
01/29/98  09:42a                   758 lcfd.bk
01/29/98  03:08p                 2,722 lcfd.log
01/29/98  09:42a                    57 lcfd.st
              13 File(s)          5,935 bytes
                           539,829,248 bytes free
```

The two LCF endpoint executables for Tivoli Distributed Monitoring, dminstall.exe, and **dogEndpoint.exe** reside in the following subdirectories under the cache:

```
C:\Tivoli\lcf\dat\01201822.138\cache\sentry\w32-ix86
C:\Tivoli\lcf\dat\01201822.138\cache\bin\w32-ix86\TME\SENTRY
```

These are the Tivoli Distributed Monitoring executables that are downloaded from the LCF gateway when distributing a Tivoli Distributed Monitoring profile.

The lcfd daemon running on an LCF endpoint logs output to the *lcfd.log* file that shows the communication between the LCF endpoint and the LCF gateway. The log can also be used for debugging purposes to check whether any errors were recorded during the LCF endpoint install phase or during an LCF endpoint profile distribution transaction. We show a sample output of the log file on the Windows NT host (wtr05099):

```
C:\Tivoli\lcf\dat\01201822.138>type lcfd.log

Jan 29 09:42:49 1 lcfd lcfd 2.1 (w32-ix86)
Jan 29 09:42:49 1 lcfd run_dir: 'C:\Tivoli\lcf\dat\01201822.138'
Jan 29 09:42:49 1 lcfd logging to 'C:\Tivoli\lcf\dat\01201822.138\\lcfd.log' at level 1
Jan 29 09:42:49 1 lcfd cache: 'C:\Tivoli\lcf\dat\01201822.138\\cache'
Jan 29 09:42:49 1 lcfd cache limit: '20480000'
Jan 29 09:42:49 1 lcfd cache size at initialization: '1313280'
Jan 29 09:42:50 1 lcfd node_login: listener addr '0.0.0.0+1227'
Jan 29 09:42:50 1 lcfd Trying last known gateway ...
Jan 29 09:42:50 1 lcfd write login file 'lcf.dat' complete
Jan 29 09:42:50 1 lcfd final pid: 99
Jan 29 09:42:50 1 lcfd Login to gateway complete.
Jan 29 09:42:50 1 lcfd Ready.  Waiting for requests (0.0.0.0+1227).
Jan 29 15:05:07 1 lcfd run_dependency, launching dependency as task
'C:\Tivoli\lcf\dat\01201822.138\cache\sentry\w32-ix86\dminstall.exe'.
Jan 29 15:05:45 1 lcfd Spawning: C:\Tivoli\lcf\dat\01201822.138\cache
\bin\w32-ix86\TME\SENTRY\dogEndpoint.exe, ses: 21b11605
Jan 29 15:05:46 1 engineUpdate Sending msg amBegPush
Jan 29 15:05:46 1 engineUpdate Sending msg amSetDatum
Jan 29 15:05:46 1 engineUpdate Sending msg amSetDatum
Jan 29 15:05:46 1 engineUpdate Sending msg amSetDatum
Jan 29 15:05:46 1 engineUpdate Sending msg amSetEnv
Jan 29 15:05:47 1 engineUpdate Sending msg amTmrRemove
Jan 29 15:05:47 1 engineUpdate Sending msg amMpeRemove
Jan 29 15:05:47 1 engineUpdate Sending msg amRaRemove
Jan 29 15:05:47 1 engineUpdate Sending msg amTmrAdd
Jan 29 15:05:47 1 engineUpdate Sending msg amMpeAdd
Jan 29 15:05:47 1 engineUpdate Sending msg amRaAdd
Jan 29 15:05:47 1 engineUpdate Sending msg amRaAddTasks
Jan 29 15:05:47 1 engineUpdate Sending msg amEndPush
Jan 29 15:08:37 1 lcfd run_dependency, launching dependency as task
'C:\Tivoli\lcf\dat\01201822.138\cache\sentry\w32-ix86\dminstall.exe'.
Jan 29 15:08:37 1 lcfd Spawning: C:\Tivoli\lcf\dat\01201822.138\cache
\bin\w32-ix86\TME\SENTRY\dogEndpoint.exe, ses: 21b1160b
Jan 29 15:08:38 1 engineUpdate Sending msg amBegPush
Jan 29 15:08:38 1 engineUpdate Sending msg amSetDatum
Jan 29 15:08:38 1 engineUpdate Sending msg amSetDatum
Jan 29 15:08:38 1 engineUpdate Sending msg amSetDatum
Jan 29 15:08:38 1 engineUpdate Sending msg amSetEnv
Jan 29 15:08:38 1 engineUpdate Sending msg amTmrRemove
Jan 29 15:08:38 1 engineUpdate Sending msg amMpeRemove
Jan 29 15:08:38 1 engineUpdate Sending msg amRaRemove
Jan 29 15:08:38 1 engineUpdate Sending msg amTmrAdd
Jan 29 15:08:38 1 engineUpdate Sending msg amMpeAdd
Jan 29 15:08:38 1 engineUpdate Sending msg amRaAdd
Jan 29 15:08:38 1 engineUpdate Sending msg amMpeDisable
Jan 29 15:08:38 1 engineUpdate Sending msg amRaAddTasks
Jan 29 15:08:38 1 engineUpdate Sending msg amEndPush
```

Figure 26. lcfd.log File

The following screen reveals the monitor file that got created on the LCF endpoint when we distributed a profile to this LCF endpoint.

```
C:\Tivoli\lcf\dat\01201822.138\.sntcfg>type monitor

{ 1 { { "NT_LogicalDisk" "FreeMegabytes" "lcfnt#lcfnt_1449767681.1.641
#Sentry::All#__0e1449767681.7.500+" } "(\"@U_lcfnt\",\"@G_lcfnt\",
\"lcfnt1449767681.7.500+\")FreeMegabytes(\"C:\")
 : \"normal\";" { 8 "TIMESTAMP=0" "RESULT=" "ARG=" "ETEXT="
 "ESTATUS=0" "RELATIVE=0" "UID=-99" "GID=-99" } null FALSE } }
```

The other files created were all related to the distribution of the profile we made to this LCF endpoint. The following screen lists the files in the .sntcfg directory:

```
C:\Tivoli\lcf\dat\01201822.138\.sntcfg>dir
 Volume in drive C has no label.
 Volume Serial Number is 407E-0541

 Directory of C:\Tivoli\lcf\dat\01201822.138\.sntcfg

02/03/98  02:28p        <DIR>          .
02/03/98  02:28p        <DIR>          ..
01/30/98  08:24a                   271 data
02/03/98  02:28p                     0 dirlist
01/30/98  08:24a                    23 eng
01/29/98  03:05p                    37 ipc
01/30/98  08:24a                   307 monitor
01/30/98  08:24a                   294 response
01/30/98  08:24a                   197 timer
               9 File(s)          1,129 bytes
                             539,842,048 bytes free
```

2.3.1.2 The Web Interface

The Web interface provides the capability to perform management tasks using a Web browser interface. The LCF endpoint includes a built-in HTTP daemon which can be accessed using your favorite Web browser. To connect, you use an URL that includes the system host name or IP address followed by the port number on which the endpoint daemon (lcfd) is listening.

Note

If you restart the LCF daemon, it will use a new port that differs from the original defined port (9494). You can have problems connecting to the LCF daemon if you don't know the new port number. You can use the wep command to get information for the endpoint:

```
 wep wtr05204
object 1056670195.11.500+
   label wtr05204
      id 8DL4FVHDQ36SVZQNDG8K0000058E
 gateway 1056670195.2.27#TMF_Gateway::Gateway#
 netload OBJECT_NIL
   crypt 0
  interp w32-ix86
 address 9.24.104.182+1060
  policy OBJECT_NIL
   httpd tivoli:zweQSX-k
   alias OBJECT_NIL
```

In this example wtr05204 is the name of our endpoint. You can see that it is using port 1060.

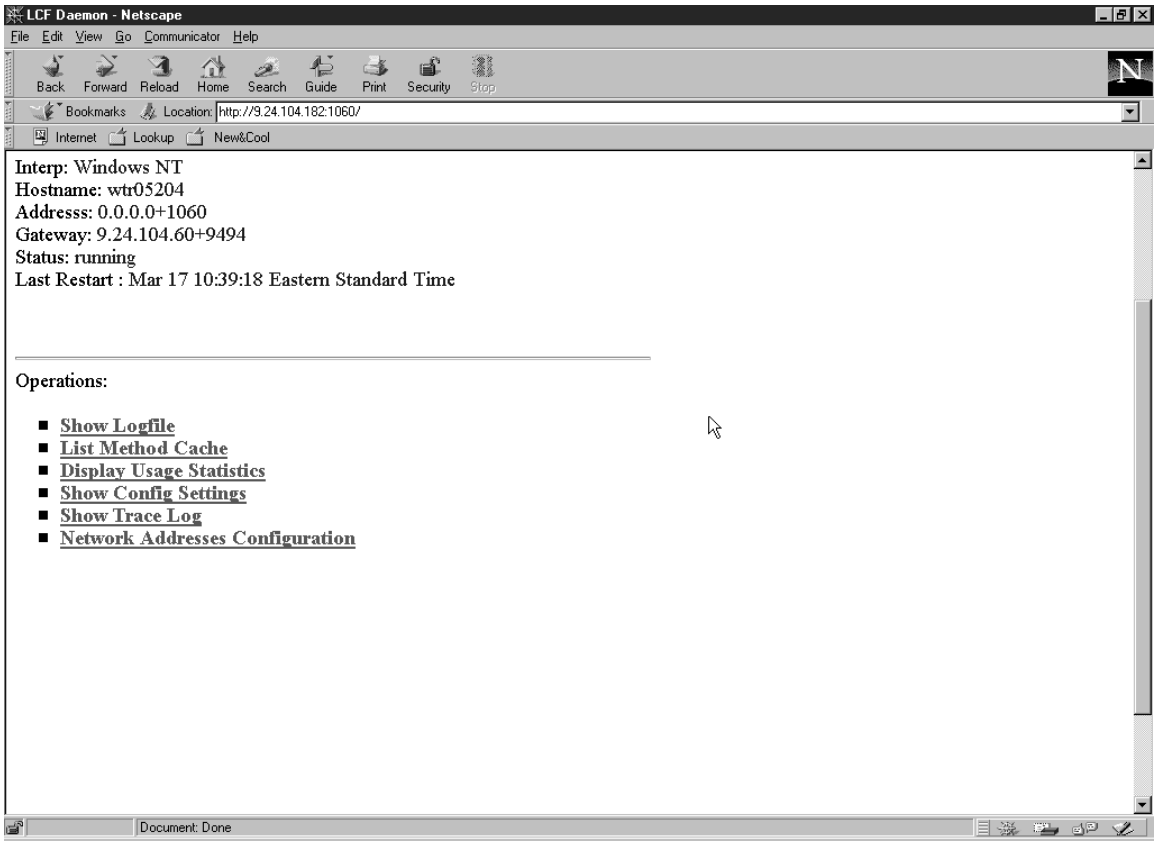


Figure 27. LCF Daemon Web Page

The above figure shows the LCF daemon status of wtr05204, which is one of our Windows NT LCF endpoints. From this interface, you can perform the following operations:

- Show Logfile

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

The Show Logfile option displays the contents of the file lcfd.log located under C:\Tivoli\lcf\dat\03171223.198.

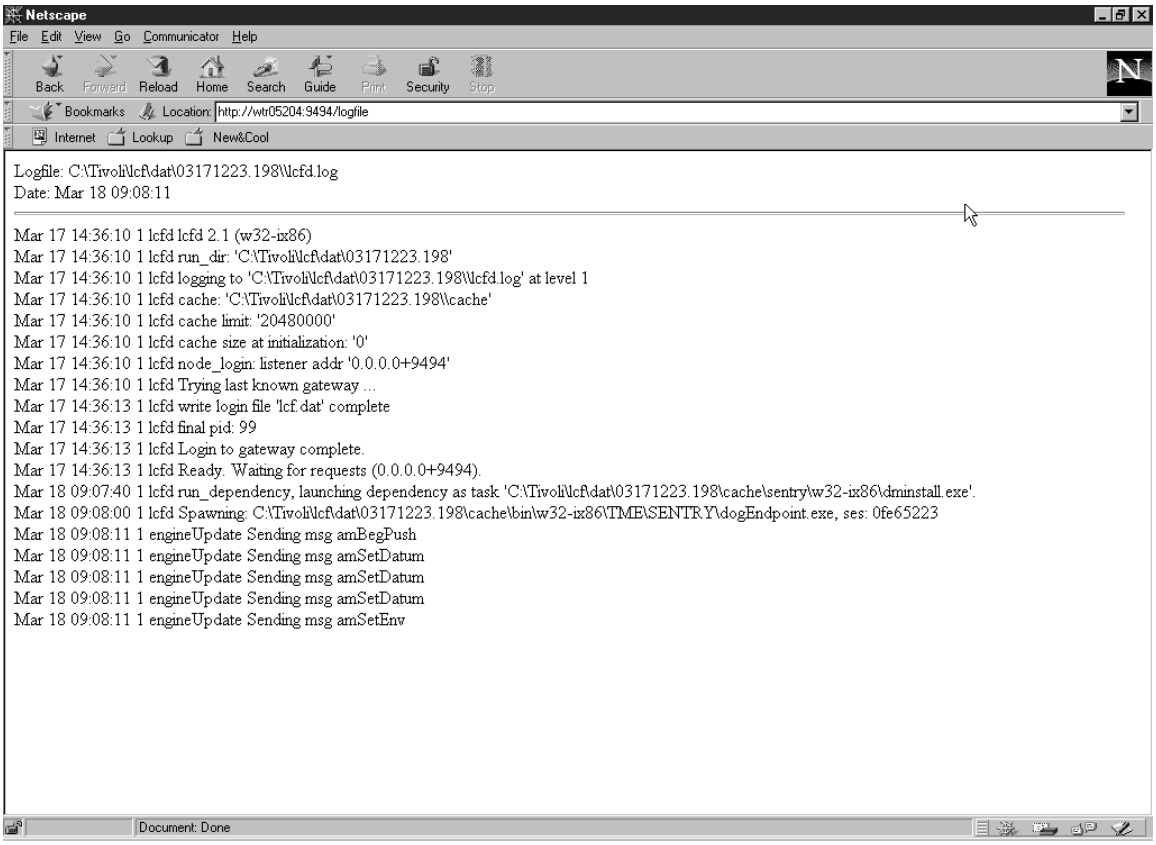
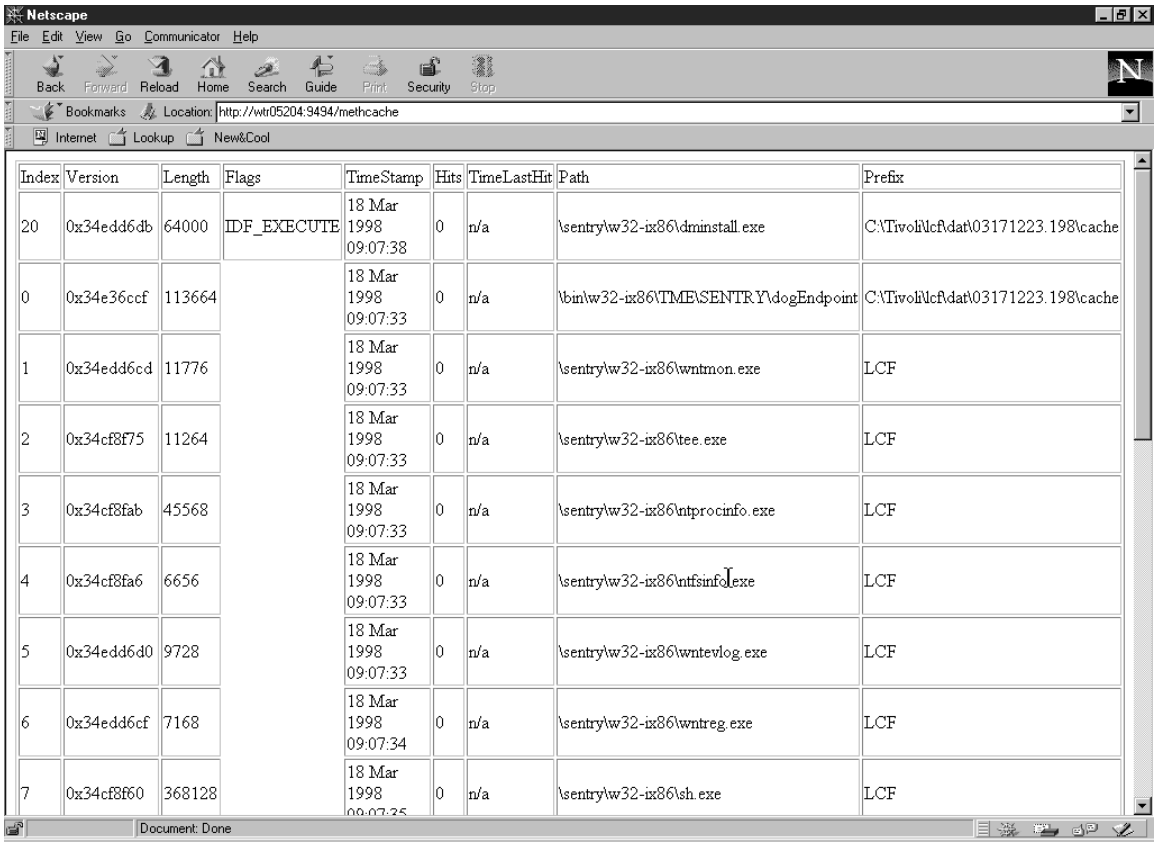


Figure 28. LCF Daemon Show Logfile Page

The List Method Cache option displays the methods that have been downloaded from the LCF gateway. We can also see these applications on the endpoint cache by exploring the directory C:\Tivoli\lcf\dat\03171223.198\cache\Lcf\Sentry\w32-ix86.



Index	Version	Length	Flags	TimeStamp	Hits	TimeLastHit	Path	Prefix
20	0x34edd6db	64000	IDF_EXECUTE	18 Mar 1998 09:07:38	0	n/a	\sentry\w32-ix86\dminstall.exe	C:\Tivoli\lcf\dat\03171223.198\cache
0	0x34e36ccf	113664		18 Mar 1998 09:07:33	0	n/a	\bin\w32-ix86\TME\SENTRY\dogEndpoint	C:\Tivoli\lcf\dat\03171223.198\cache
1	0x34edd6cd	11776		18 Mar 1998 09:07:33	0	n/a	\sentry\w32-ix86\wntmon.exe	LCF
2	0x34cf8f75	11264		18 Mar 1998 09:07:33	0	n/a	\sentry\w32-ix86\tee.exe	LCF
3	0x34cf8fab	45568		18 Mar 1998 09:07:33	0	n/a	\sentry\w32-ix86\ntprocinfo.exe	LCF
4	0x34cf8fa6	6656		18 Mar 1998 09:07:33	0	n/a	\sentry\w32-ix86\ntfsinfo.exe	LCF
5	0x34edd6d0	9728		18 Mar 1998 09:07:33	0	n/a	\sentry\w32-ix86\wntevlog.exe	LCF
6	0x34edd6cf	7168		18 Mar 1998 09:07:34	0	n/a	\sentry\w32-ix86\wntreg.exe	LCF
7	0x34cf8f60	368128		18 Mar 1998 09:07:35	0	n/a	\sentry\w32-ix86\sh.exe	LCF

Figure 29. LCF Daemon List Method Cache

You can see that the Tivoli Distributed Monitoring executables, dminstall.exe and dogEndpoint.exe, have been downloaded to the LCF cache of the Windows NT endpoint.

The w32-ix86 that you can see in the path of both files indicates that these are the method implementations for Windows NT.

This option in the LCF Daemon http page can be useful to look at when you have problems distributing or running a monitor on an LCF endpoint.

Chapter 3. Custom Monitoring Examples

This chapter provides examples of custom monitors. The monitors were created to supplement the monitors already available in TME 10 Distributed Monitoring 3.6.

We developed these monitors in response to experiences that we have encountered in our practical work life. The intention of this chapter is to convey to the user the process of developing custom monitors using MCSL. The monitors described in this chapter could all use the numeric or string monitor to drive them, but using MCSL has many advantages over using generic script monitors.

These advantages are:

- 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 used to support multiple systems.
- The monitor is portable and can be easily installed on another TME system.
- Since the template definition of a monitor does not change significantly between two monitors, code can be reused.

The monitoring examples included in this chapter accomplish the following:

- The first example shows how we can capture summary statistics from the UNIX `ping` command to show the packet loss percent for a specific node in the network. Alternatively, the same monitor can be used with a different parameter to capture the time that it takes for an ICMP echo to reach the destination node and back. This allows the user to determine how fast or slow the network link is during a specified time interval that the user may be interested in. This monitor is described in 3.4, “Packet Loss and Delay Using UNIX Ping” on page 40.
- The second example queries the default domain name server to determine whether it can resolve the IP or name query submitted to it successfully. This monitor can be used to determine whether the default domain name server is listening on the default port (53) and responding to queries. Many applications, including Tivoli, depend on the domain name server being alive and responding correctly. This monitor is defined in 3.5, “Query Response From a Name Server” on page 53.
- The next example is a monitor that can query whether an application is actively accepting connections on a specified port. It could be the port of a well-known service such as FTP or HTTPD but you could also specify any application port that a user application opens to service requests. This monitor can be used to determine whether the user application is active on a specified host. This monitor is defined in 3.6, “Query Well-Known Service Application Ports” on page 60.
- Another example shows how we can use an SNMP query to retrieve the MIB-II value from an IBM2210 router. This monitor is described in 3.7, “Using SNMP Query To Determine Router Statistics” on page 67.
- The next example is a generic log file parser. This monitor can be attached to any log file that needs to be monitored for a specific field pattern. TME 10 Distributed Monitoring 3.6 has a file search monitor already that searches for

specific patterns, but this monitor complements the existing one, by searching on a field-by-field basis. This can be used, for example, to monitor the syslog or security.log file on an AIX system for specific field values. This monitor is defined in 3.8, “Generic Log File Parser” on page 78.

- In 3.9, “Monitor to Download a Web Page” on page 90 we describe a monitor that downloads a Web page and returns either the size of the Web page in bytes or the time to download the Web page. This monitor can be used, for example, to check regularly if a Web service is available and how it performs.
- In 3.10, “Monitoring an IBM 8235” on page 99 we create a monitor for an IBM 8235, which is a hardware device for remote LAN access. In order to be able to monitor this device, we create a monitor that performs an automated telnet. This is achieved using the Expect tool.
- We use the Expect tool again in 3.11, “Monitoring Performance of a VM System” on page 111 where we create a monitor for various performance indicators of VM, an IBM mainframe operating system that is not directly supported by Tivoli. This is also an example of how you can monitor operating systems even if they are not directly supported by the Tivoli framework.
- In 3.12, “Monitoring DB2” on page 124 we show how to create a monitoring collection for the DB2 database server. We give examples of how to extract data from the DB2 server and how to include this data in Tivoli Distributed Monitoring. This monitor can be taken as an example of a general approach to writing monitors for an RDBMS.

  In 3.13, “Monitoring File Systems” on page 152 we present a monitor for file system usage that can monitor several file systems at the same time and also allows you to use wildcards when monitoring file systems. This monitor is available for AIX, Solaris and HP-UX. We also integrate this monitor with TEC.

3.1 Before You Start

Before starting to write your monitoring collection, there are a few things to keep in mind:

1. Naming convention

   The monitor name can only contain alphabetic characters. The compiler will not give any errors if you use a non-alphabetic character in the monitor name, but when you distribute your profile you'll get the following message in the SentryStatus notices:

   ```
   expected monitor primitive
   ```

2. Changing a monitoring collection

   If you change an existing monitoring collection that is being used in a profile, you will destroy that profile. It will not be possible to modify the profile. There is only one way out - delete the profile and start all over.

3. Deleting a monitoring collection

   The `mcsl` command does not have a delete option. To delete a monitoring collection, execute the following command:

   ```
   wlookup -r MonitoringCapabilityCollection -a
   ```

   This command will display all installed monitoring collections. Select from the list the one you want to delete and issue the next command.

```
wdel @MonitoringCapabilityCollection:monitor_collection_name
```

3.2 Before You Start Writing MCSL Scripts

To create a monitoring collection and compile it using MCSL you need to have the following environment in place:

- Tivoli Management Platform, Version 2.0.2 or later
- Tivoli/Sentry 2.0.2 or later or TME 10 Distributed Monitoring 3.6
- A C language preprocessor

  On AIX you need to have a valid license key or you can copy the cpp command binary from an AIX 3.2.5 system, which will work without a license key.

- Access the following files:

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

  These are include files that define message and data structures.

  Note

  All the files above are now included with TME 10 Distributed Monitoring 3.6, except gencmsg which comes as part of the Tivoli ADE package. You will have to install the ADE package to invoke this file. ADE comes with the Tivoli Framework.

3.3 Overview of Creating A Monitoring Collection

There are six steps in creating a monitoring collection with MCSL.

Note

The following steps just describe how to create a monitoring collection, that is, linking the monitors to Tivoli. They do not involve the actual implementation of the monitoring code, which is usually a shell script or a C program.

1. Create the message catalog.

   MCSL refers to messages using a label (tag), instead of directly hardcoding these messages in the monitor definition file. One of the advantages of this approach is that you can keep the same labels in the monitor definition file but translate them into different languages based on the locale in the message catalog file.

2. Create the monitor definition file.

   The monitor definition file is the heart of the monitor that you define. This file defines the name of the monitoring collection, and the monitors in it. It will specify the arguments that will be passed to a monitor script, as well as the name of the script to be used with the monitor. The scripts can be included directly in the definition file or be referenced by including the name of the shell

script. GUI elements, such as selection lists and data entry fields, are defined in this definition file. The definition file has an extension of `.csl`.

3. Compile the message catalog file.

   The message catalog file has a format of:

   ```
   $ label
   # text
   ```

   which are message definitions for the labels that you will use in the monitor definition file. This file is translated by the C preprocessor using the gencmsg command. The argument to the gencmsg command is the name of the message catalog file with the `.msg` extension.

4. Compile the monitor definition file.

   After the message catalog file has been run through the C preprocessor, you will include the `.dsl` file produced by it in your monitor definition file. The monitor definition file also includes some of the other include files mentioned in 3.2, “Before You Start Writing MCSL Scripts” on page 39.

   You compile the monitor definition file using the mcsl command. The result of the compilation will be a file with the same name as your monitor definition file but with a `.col` extension. This is the collection file that can be distributed.

5. Install or distribute the new monitoring collection.

   This final step again uses the mcsl command to install the new monitoring collection in all the running TME nodes.

6. Restart the Object Dispatchers.

   To see your new monitoring collection in the list of available monitors, you will have to stop and start all the object dispatchers on the running TME nodes.

3.4 Packet Loss and Delay Using UNIX Ping

Our first monitor implements a special kind of ping request. It checks if a host is available and can also return the roundtrip time for the ICMP request.

We start with this rather simple example to explain the process of creating a monitoring collection in detail and then show some more sophisticated monitoring examples in the subsequent sections.

3.4.1 Monitor Overview

The ping command is normally supported in all flavors of TCP/IP. It is used to determine whether a host on a network is reachable or not. This is true for most of the scenarios except in cases where we are trying to reach a host beyond a firewall that is blocking ICMP requests.

The ping command can also be used to determine (roughly) whether there is a significant delay in reaching the host, by computing the average round-trip time that it takes for the ICMP echo request to reach the host and back.

This monitor extracts the time value from the response from the ping command and computes an average, which is the output from the script. A monitoring threshold can be set in TME 10 Distributed Monitoring 3.6 to send a notice about a slow network link to an administrator.

This monitor also accepts as an argument a packet loss parameter. If this argument is supplied it will extract the summary packet loss percentage figure that is displayed in a ping command as output. We may want to monitor the packet loss percentage and inform a network administrator about consistent packet loss percentages greater than a specified threshold.

The actual shell script that implements the monitor is shown and explained in 3.4.9, “Source Script Used in the Custom Ping Monitor” on page 51. We first explain the necessary MCSL definitions and then explain the monitoring code itself.

3.4.2 Creating the Message Catalog

The message catalog is defined in a .msg file (usrdelay.msg in our example). Each entry in this message file corresponds to the following format:

```
$ key=keyname
# message_text
```

The keyname field is a label (tag) that is used to access the message from the monitor definition file. The pound or hash (#) symbol is a unique message number that contains the message text to be displayed. You need to have an entry for each text item that will appear in the user interface, such as help text messages, field descriptions for command arguments, selection list entries and button choices. Figure 30 shows the userdelay.msg file that we created for our sample monitor:

```
$ key=About_MonCol
1 This Monitor will return a Numeric Value based on the option
selected by the user. There are two option choices, delay will
return the average round trip needed for the ICMP echo request,
and packetloss will return the packet loss percent.
$ key=MonSrc_Descr1
2 Retrieve Ping Status
$ key=MonSrc_val_Descr1
3 (Result)
$ key=About_MonSrc_Descr1
4 This monitor will return the average delay (in millsecs) of reaching
a host on a network if the delay parameter is selected or will return
the packet loss percentage if the packetloss parameter is selected.
$ key=ButtonLabelChoice
5 Choices
$ key=choice_delay
6 delay
$ key=choice_packetloss
7 packetloss
$ key=ping_options
8 Options
$ key=host_address
9 IPaddress
```

Figure 30. Userdelay Message Catalog File

3.4.3 Compiling the Message Catalog

Once you have created the userdelay.msg file, you have to translate the file into a format that can be included in the monitor definition. Use the gencmsg command to do that:

```
gencmsg userdelay.msg
```

Three new files will be created:

```
userdelay.c
userdelay.h
userdelay.dsl
```

The userdelay.dsl file will be included as a header line in the monitor definition file (userdelay.csl) as shown in the next step. The following is an output from the gencmsg command:

```
root@rs600019:/:/project/distmon/rbanerje/custom# gencmsg userdelay.msg
gencmsg: creating ./userdelay.h
gencmsg: creating ./userdelay.dsl
gencmsg: creating ./userdelay.c
root@rs600019:/:/project/distmon/rbanerje/custom#
```

Figure 31. Output from the gencmsg Command

3.4.4 Compiling the Monitor Definition File

The monitor definition is in a file with an extension of .csl. In our example we call it userdelay.csl. Figure 32 on page 43 shows the contents of userdelay.csl, with notes about each section.

```
#include "Sentry2_0.dsl"
#include "userdelay.dsl"

Collection "CustomPing" {
 CodeID = "$Id:userdelay.csl,v 1.0$";
 Version = "1.0";
 Require = ">2.0.2";
 HelpMessage = (userdelay_About_MonCol);
 EventBaseClass = "Sample_Sentry_Monitors";
 NoticeGroup = "Sentry";

#include "choicelists.csl"
#include "operators.csl"
#include "formats.csl"

 ChoiceList DiffOptions {
  ButtonLabel = (userdelay_ButtonLabelChoice);
  {
   { (userdelay_choice_delay) "delay" }
   { (userdelay_choice_packetloss) "packetloss" }
  };
 };

 Monitor CustomPing Numeric Group numeric {
 Description = (userdelay_MonSrc_Descr1);
 ValueDescription = (userdelay_MonSrc_val_Descr1);
 HelpMessage = (userdelay_About_MonSrc_Descr1);
 Argument ( userdelay_host_address)
 DefaultValue "127.0.0.1";
 Argument(userdelay_ping_options)
   RestrictedChoice "DiffOptions"
   DefaultValue "delay";

 Implementation ( aix3-r2, aix4-r1, sunos4, solaris, hpux9, hpux10)
 Shell("/bin/sh", "-c", Command, "USR_PING")
 Import "userdelay.sh"
 ;
 };

}
```

Figure 32. Userdelay Monitor Definition File

Each time we refer to a message created in the userdelay.msg catalog file, we use the keyname (label) preceded by the filename of the catalog file and an underscore. For example, if we want to refer to the About_MonSrc label that we defined in the message catalog, we refer to it as userdelay_About_MonSrc in the monitor definition file.

The keyword Numeric on the monitor definition specifies that this monitor is going to return a numeric value. We could also have returned a string return value or an async return value. The text `Group` in the monitor definition file is a keyword, and finally `numeric` is a reference to a definition in the operators.csl file. This controls which trigger options you have for the monitor as seen in Figure 37 on page 50. There are two options 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 monitor definition file (userdelay.csl) has a single implementation for AIX, SunOS and HP-UX. This is because the monitor script will determine which UNIX system it is running on and then execute the appropriate command.

The command has slightly different parameters for the three flavors of UNIX. A copy of the shell script used in this example is included in Figure 39 on page 51.

The `Argument` keyword, defined in the monitor definition file will, determine the arguments that get passed to the monitoring script. The monitor definition file defines two arguments, one of which is an IP address that defaults to the loopback address when the monitor is selected by the user. The second argument is an option selected from a restricted choice listbox. The options can be either `delay` or `packetloss`. Depending on the choice selected, the monitor script will either gather packet loss data or average round trip time data. It is possible to imbed the entire script in the monitor definition file, instead of using the `Import` keyword. If you do this, then each line of the script must be prefixed with a period (".").

Note

You can only imbed shell scripts directly in the MCSL definition file.

3.4.5 Compiling the Monitoring Collection

We are now ready to compile our monitor definition file. Use the following command to do this:

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

Where -P specifies the path to the preprocessor to be used, -x specifies the output file and the source .csl file. The result of this compilation is a binary file, userdelay.col, which contains the complete monitor definition, including the implementation scripts.

Note

The .col format that is produced by the mcsl command is also the format in which the monitoring collections that come with Tivoli Distributed Monitoring are shipped.

The following is the output when you run the mcsl command as defined above:

```
root@rs600019:/:/project/distmon/rbanerje/custom# c userdelay
Running mcsl with the following substitution
mcsl -P ./cpp -x ./userdelay.col userdelay.csl -lang-c++
-nostdinc -undef
cpp: cannot retrieve message number 1501-232
cpp: cannot retrieve message number 1501-232
cpp: cannot retrieve message number 1501-232
Compilation successful
```

Note

We created a script file called "c"(for compile) which took the name of the .csl file as an argument. This made it much easier to compile, instead of typing out the long sequence of commands at the command line. The compile phase will determine syntax errors and check to see whether the monitoring script or binary is in the path specified by the Import keyword. The compilation will only be successful if all the syntax errors are corrected and Import references are resolved.

Since we copied the AIX 3.2.5 preprocessor and ran it on an AIX 4.2 system, there were error messages displayed by the preprocessor that were unresolved. This does not affect the compilation of the monitor definition file but is because the preprocessor is missing a system message file.

3.4.6 Installing the Monitoring Collection

Once you have your userdelay.col file, you are ready to load the new monitoring collection. This needs to be done only on the TMR server. Use the following command to load the new monitoring collection:

```
mcsl -i -R userdelay.col
```

There is no output from this command to signify that the collection was installed successfully. In some cases the command will give an error message if the destination object dispatcher was not available. This is not necessarily a serious problem. The command attempts to update all the active nodes connected to the TMR server and will give this error message if some specific nodes are down.

3.4.7 Restarting the Object Dispatcher

To see the new collection and its monitors in the TME 10 Distributed Monitoring 3.6 user interface, we have to recycle the object dispatchers on all the connected nodes. You can do this by using the following command:

```
odadmin reexec all
```

Note

However, we found that at times most of the object dispatchers failed to start on the connected nodes. The `oadadmin shutdown all` and the `odadmin start all` commands were more successful in restarting all the remote connected object dispatchers.

3.4.8 Using the New Monitoring Collection

The final step is to start using the new monitoring collection. Create a TME 10 Distributed Monitoring 3.6 profile called Custom_Monitor and then double-click to open this profile. You will see the following window.

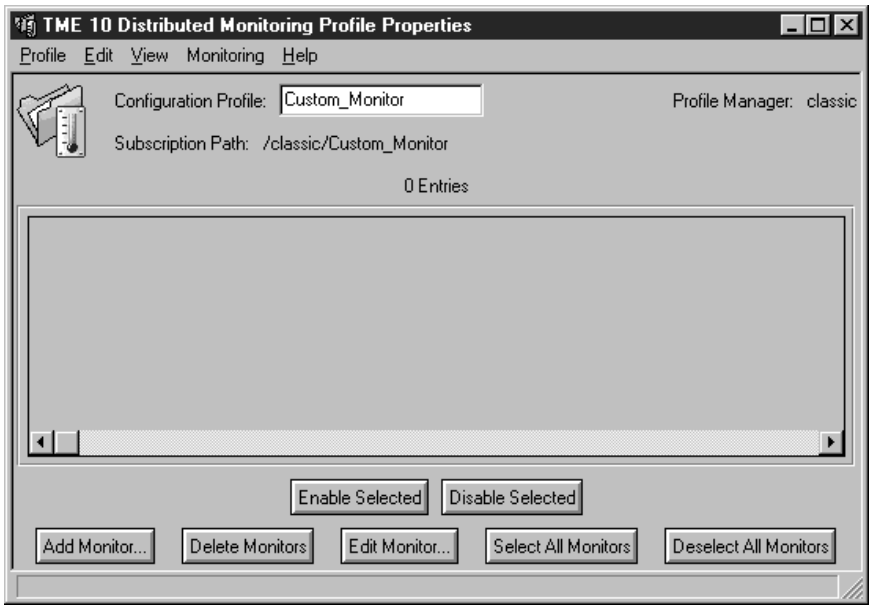


Figure 33. TME 10 Distributed Monitoring Profile Properties Window

Select **Add Monitor...** in this window.

You will then be presented with the following window:

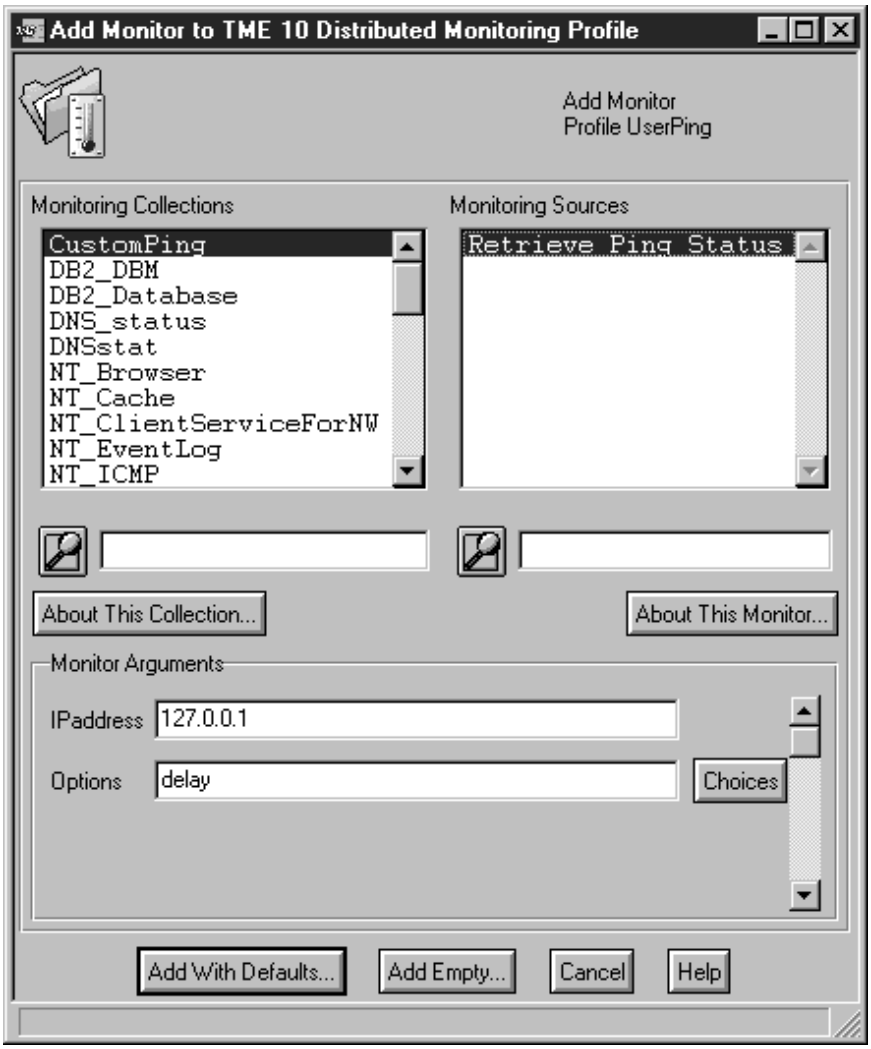


Figure 34. Add Monitor to TME 10 Distributed Monitoring Profile Window

In the left table the new monitoring collection called CustomPing will appear. You will recognize that the names that show up are the ones used in the .msg and .csl files. When you select it, the monitor that you defined, Retrieve Ping Status, will be listed on the right-hand table.

If you select **CustomPing** from the Monitoring Collections section and then click on **About this Monitor...** the display shown below will appear.

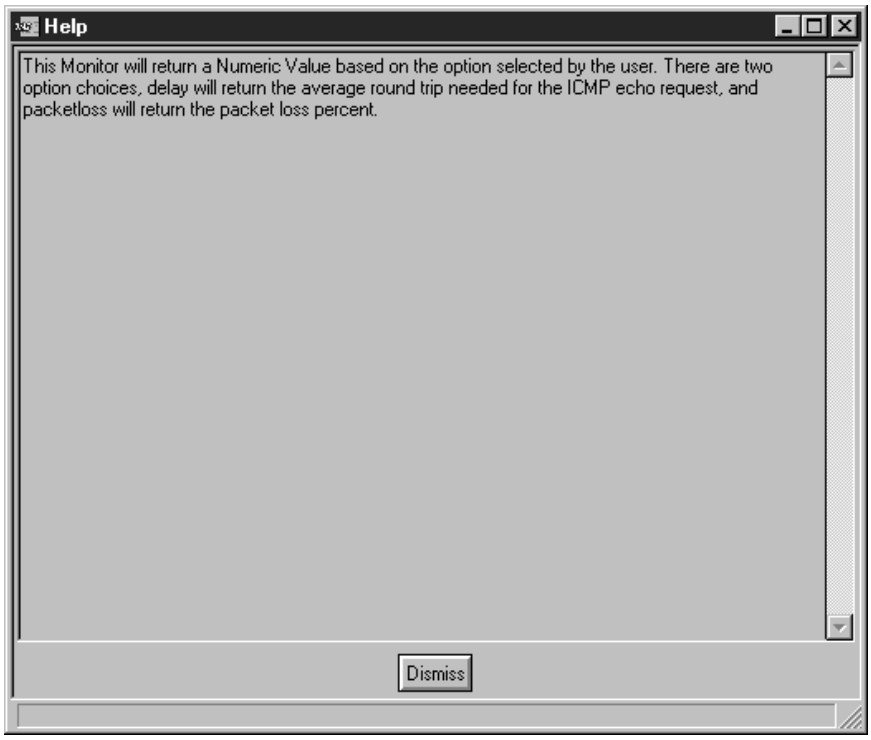


Figure 35. About Monitoring Collection Description

Notice that this is the content of the About_MonCol message catalog entry that we defined in our userdelay.msg file (see Figure 30 on page 41). It appears here because we specified the follwing in our monitor definition file (see Figure 32 on page 43):

```
HelpMessage = (userdelay_About_MonCol)
```

Click on **Dismiss** to close the window and return to the monitoring collection window. Select **Retrieve Ping Status** from the right-hand table. You will see the Options and the IP address textbox being displayed in the lower half of the monitoring collection window as shown in Figure 34 on page 47.

Add the host IP address that needs to be "pinged" in the IP address textbox. The default value shows as 127.0.0.1. Now click on **Choices** beside the Options field to get the Ping status values that you can retrieve. You are presented with the following window:

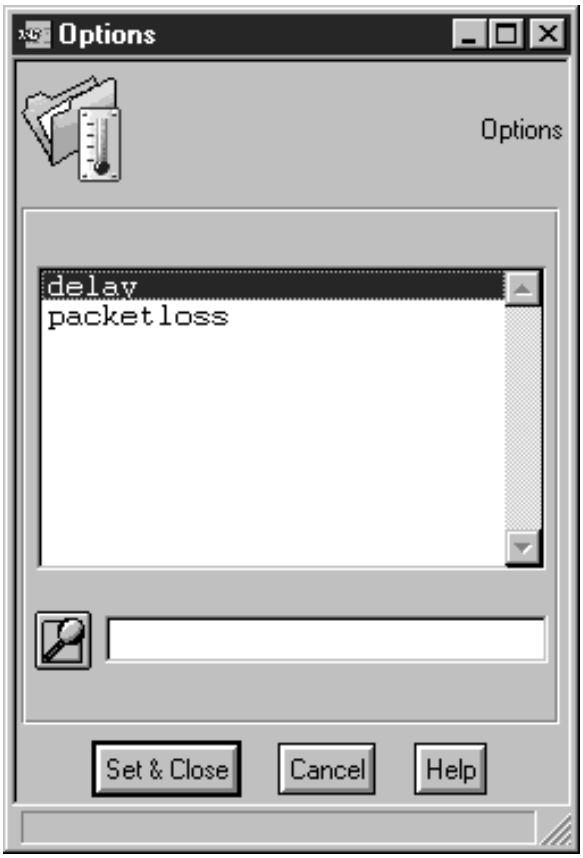


Figure 36. Options for Custom Ping Monitor

Select a value from the list and then press the **Set & Close** button. This will put the selected value in the Options text field. For example, selecting **delay** will put `delay` in the textbox.

After adding the required parameters for the monitor, click on **Add Empty...**. You are then presented with the normal window for editing the monitor details, as shown in the following figure.

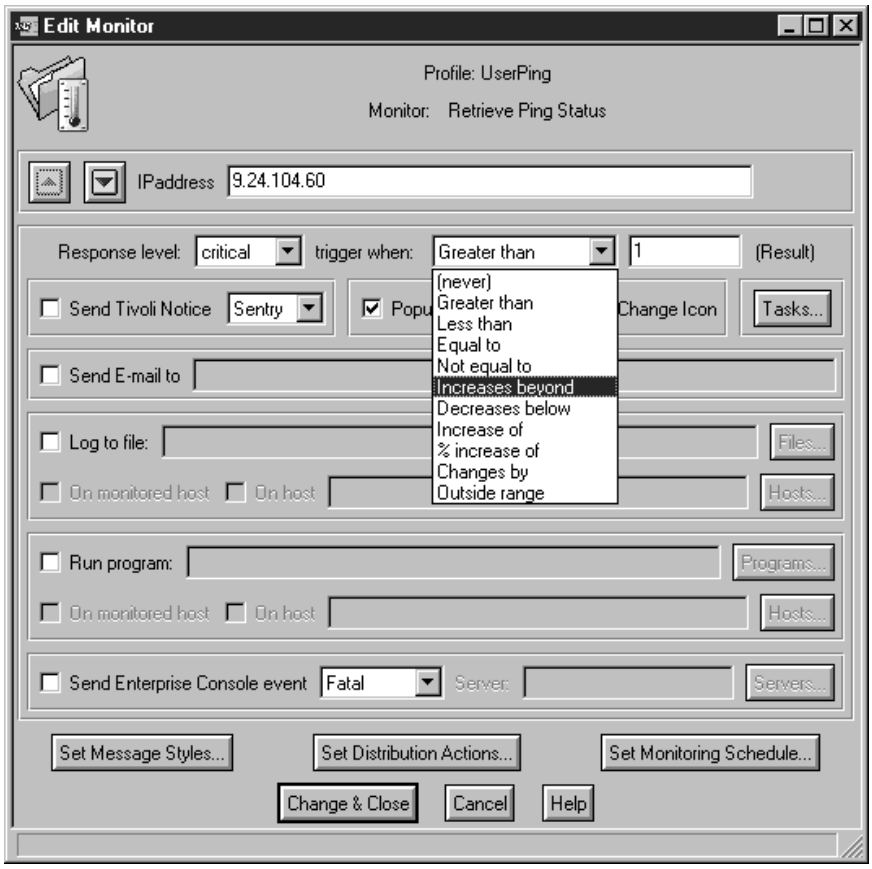


Figure 37. Edit Monitor Window

For test purposes, let's select a critical response for this monitor to trigger when it has a value greater than 1 millisecond. Click on **Change & Close** when you have finished. The end result is that if the delay value retrieved from pinging a node is greater than the threshold value of 1 millsecond, a pop-up message will be displayed to the Root administrator in policy region rs600019.

This is shown in Figure 38.

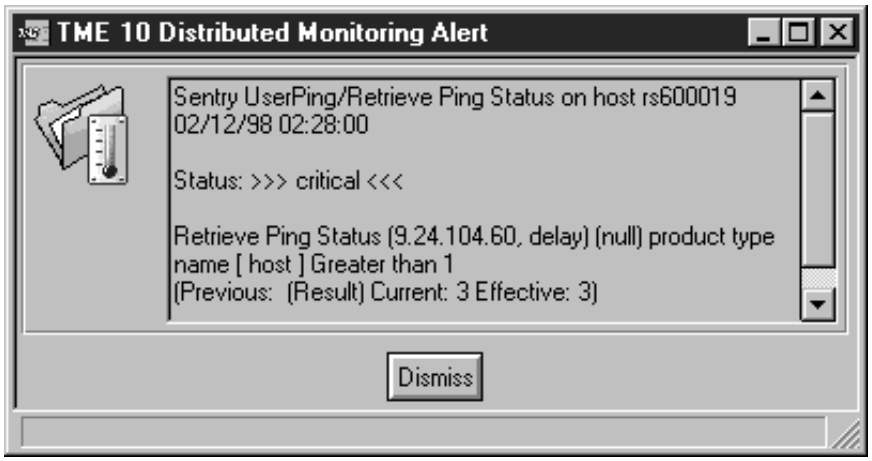


Figure 38. Alert from Custom Ping Monitor

In the above alert the delay is shown as 3 milliseconds.

3.4.9 Source Script Used in the Custom Ping Monitor

The source code for the shell script used in the user ping example is included in this section. This is the actual monitor code.

```
#!/bin/ksh
#####################################################################
#
# FILE:         userdelay.sh
# VERSION:      1.00 22 January 1998
# AUTHOR:       Robi Banerjee, rob9@us.ibm.com
#
# COPYRIGHT:
#
#   All Rights Reserved
#   Licensed Materials - Property of IBM
#
# This script was designed to be run from Tivoli Sentry 3.2 to
# provide a status of how long (time in millisecs) it takes for
# a ICMP echo request to reach a host and the host to acknowledge
# back. This script can be run from a TMR server, a Managed Node or
# an LCF endpoint.
#
# Shell has been tested on AIX 4.2.0 .
#
#
#####################################################################
#
# check for parameters being passed in
if (( $# < 2 ))
then
 echo "Usage: $0 [Host_ip_address] [
delay/packetloss]"
 exit 0
fi

#set the default packet count to 3
pkt_count=3

# set the path to the ping program
export PATH=$PATH:/usr/sbin:/usr/bin
```

Figure 39. Source Code for userdelay.sh (Part 1 of 3)

The first part of the monitor script checks for the correct number of arguments and then sets a default packet count and the program path.

```
# get the IP address of the hostname that needs to be checked
host_ip=$1
option=$2
#
system= uname
case $system in

  AIX)

 if [[ $option = "delay" ]]; then
  delay_expr= ping -c 3 $host_ip | grep time | sed 's/time=//'
| awk '{printf "%s + 0",$7}' ;
  echo $delay_expr;
 else if [[ $option = "packetloss" ]]
; then
  lost_percent= ping -c $pkt_count $host_ip | grep "%" |
awk  '{print substr($7,1,length($7)-1)}' ;
  echo $lost_percent;
  fi
 fi
 exit 0
  ;;

  SunOS)

 if  [[ $option = "delay" [[; then
  delay_expr= ping -s $host_ip -c $pkt_count | grep time |
sed 's/time=//' | awk '{printf "%s + 0",$7}' ;
  echo $delay_expr;
 else if [[ $option = "packetloss" ]]
; then
  lost_percent= ping -s $host_ip -c $pkt_count | grep "%" |
awk  '{print substr($7,1,length($7)-1)}' ;
  echo $lost_percent;
  fi
 fi
 exit 0
  ;;
```

Figure 40. Source Code for userdelay.sh (Part 2 of 3)

As there are some differences between the different flavors of UNIX, two implementations are shown, one for AIX and one for Solaris. The operating system is determined using the uname command. The ping command is invoked and the result is then parsed to extract the information we want. The source code continues in Figure 41 on page 53.

```
  HP-UX)

 if [[ $option = "delay" ]]; then
  delay_expr= ping $host_ip 64 $pkt_count | grep time |
sed 's/time=//' | awk '{printf "%s + 0",substr($6,1,length($6)-1)}' ;
  echo $delay_expr;
 else if [[$option = "packetloss"]]
; then
  lost_percent= ping $host_ip 64 $pkt_count | grep "%" |
awk  '{print substr($7,1,length($7)-1)}' ;
  echo $lost_percent;
  fi
 fi
 exit 0
  ;;
  *)
 print "# $0 Operating system not recognized"
 exit -1;;
esac

#check to see whether we have captured any time data, if not
#the host probably could not be reached
if  [[$delay_expr = ""]]
;then
{
 echo -1
 exit 0
}
fi
#invoke the bc command and pass in the data
bc <<EOF
s=$delay_expr
s / 3
quit

EOF
```

Figure 41. Source Code for userdelay.sh (Part 3 of 3)

Above, another implementation is shown, this time for HP-UX. At the end of the script, the result is written to stdout and the monitor exits with a return code of 0.

3.5 Query Response From a Name Server

This sample monitor is also written using a Korn shell and then wrapped using MCSL and can be used from a TME 10 Distributed Monitoring 3.6 profile to find out the status of the default name server. The default name server on UNIX systems is normally defined in the /etc/resolv.conf file. The user can also modify the script, if a server other than the default is to be used for the nslookup response.

3.5.1 Monitor Overview

This monitor queries the default name server on your host (defined in the /etc/resolv.conf file) with either an IP address or a fully qualified hostname. The monitor then intercepts the response sent from the name server and will output a success or failure message depending on whether the name server resolved the query successfully.

We created this monitor using a simple shell script that executes the nslookup command that is available on most UNIX and Windows NT systems. The output from the nslookup command is then analyzed to see whether it matches the input from our query, which could have been either an IP address or a fully qualified hostname. If the output from the nslookup command matches our query parameter, we indicate that the name server resolved our query successfully.

A copy of this script is included in Figure 46 on page 59.

We could have implemented this same monitor using the monitor shown in 3.6, "Query Well-Known Service Application Ports" on page 60 to check whether the name server is listening on its default port 53. However, in addition to checking whether the application is listening on a port, our query successfully tells us whether the name server is responding correctly.

3.5.2 Creating the Message Catalog

We create the monitoring collection using MCSL as we did before in our custom ping example shown in 3.4, "Packet Loss and Delay Using UNIX Ping" on page 40. We will not explain the details of the MCSL syntax again here. We call our message catalog for this example dnsstat.msg.

The message catalog file for this sample monitor is shown below:

```
$ key=About_MonCol
1 This Monitor will return a Success String Value based on whether
it could query the default domain name server successfully. It
will accept as a parameter either an IP address or a fully qualified
hostname. If the hostname is partial then the search is restricted
to the default domain.
$ key=MonSrc_Descr1
2 Query Nameserver
$ key=MonSrc_val_Descr1
3 (Result String)
$ key=About_MonSrc_Descr1
4 This monitor accepts as argument an IP address or a fully qualified
hostname .It will query the default name server and return a Success
or Failure based on the result returned from the name server.
$ key=query_value
5 QueryValue(IPAddress or Name)
```

Figure 42. Message Catalog File dnsstat.msg

3.5.3 Compiling the Message Catalog

We compile the message catalog (dnsstat.msg) file using the command:

```
gencmsg dnsstat.msg
```

We then have three files, dnstat.h, dnsstat.c and dnsstat.dsl.

Remember

We will be including the dnsstat.dsl file in an `#include` statement when we create the monitor definition file in the next section.

3.5.4 Creating the Monitor Definition File

We create the monitor definition file that we will use in this example. Figure 43 shows the monitor definition file we created:

```
#include "Sentry2_0.dsl"
#include "dnsstat.dsl"
Collection "DNS_status" {
 CodeID = "$Id:dnsstat.csl,v 1.0$";
 Version = "1.0";
 Require = ">2.0.2";
 HelpMessage = (dnsstat_About_MonCol);
 EventBaseClass = "Sample_Sentry_Monitors";
 NoticeGroup = "Sentry";

#include "operators.csl"
#include "formats.csl"

 Monitor checkdns String Group string {
 Description = (dnsstat_MonSrc_Descr1);
 ValueDescription = (dnsstat_MonSrc_val_Descr1);
 HelpMessage = (dnsstat_About_MonSrc_Descr1);
 Argument (dnsstat_query_value)
  DefaultValue "";

 Implementation ( aix3-r2, aix4-r1, sunos4, solaris, hpux9, hpux10)
 Shell("/bin/sh", "-c", Command, "DNSstat")
 Import "checkdns.sh"
 ;
 };

}
```

Figure 43. Sample dnsstat.csl File

You can see that the monitor invokes the script checkdns.sh which is the actual implementation of that monitor.

3.5.5 Compiling the Monitor Definition File

We are now ready to compile our monitor definition file using the command that we described in 3.4.4, “Compiling the Monitor Definition File” on page 42. Once we execute the command we should get the following output:

```
root@rs600019:/:/project/distmon/rbanerje/custom# c dnsstat
Running mcsl with the following substitution
mcsl -P ./cpp -x ./dnsstat.col dnsstat.csl -lang-c++
-nostdinc -undef
cpp: cannot retrieve message number 1501-232
cpp: cannot retrieve message number 1501-232
cpp: cannot retrieve message number 1501-232
Compilation successful
root@rs600019:/:/project/distmon/rbanerje/custom#
```

We should now have a dnsstat.col file, which is the monitoring collection binary.

3.5.6 Installing the Monitoring Collection

The monitoring collection can now be installed using the command:

```
mcsl -i -R dnsstat.col
```

This command will install the new monitoring collection on the TMR server.

3.5.7 Restarting the Object Dispatcher

To see the new monitoring collection in the TME 10 Distributed Monitoring 3.6 user interface, you will have to restart the object dispatchers on all the connecting nodes using the command:

```
odadmin reexec all
```

3.5.8 Using the New Monitoring Collection

The final step is to start using the new monitoring collection. We can add the new monitor to our Custom_Monitor profile that we created in our previous example for custom ping in Figure 33 on page 46. A snapshot of the new monitoring collection we created in this example is seen when we select **Add Monitor...** (Figure 44 on page 57.).

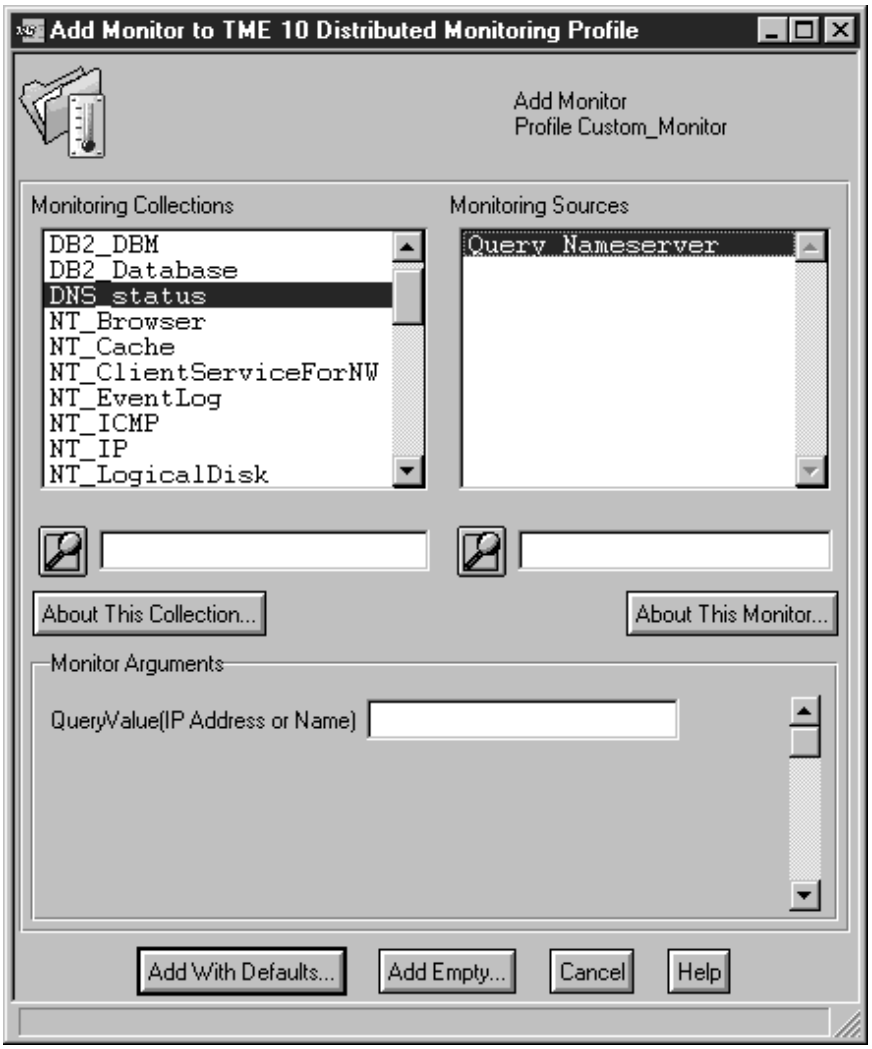


Figure 44. Add Monitor to TME 10 Distributed Monitoring Profile Window

For test purposes, we enter the value of an IP address that we know will resolve correctly from the name server and click on **Change & Close**. The result of all this is that, if the name server responds correctly then a pop-up window is displayed at the Tivoli desktop of the administrator Root@rs600019-region where we distributed this profile. We selected a response level of `Always` in this example, but since the script returns either a "Success" or "Failure" message we could have tested for the specific string output. The following figure shows the alert message:

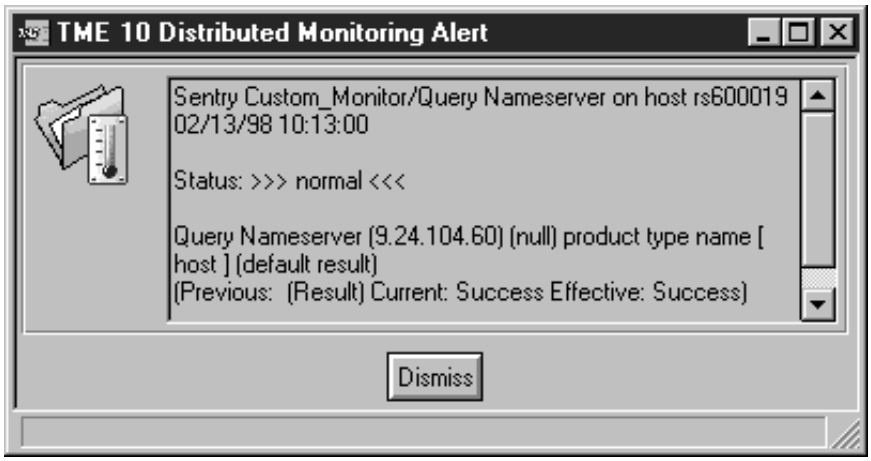


Figure 45. Alert from DNS_Status Monitor

3.5.9 Implementation Shell Script for This Monitor

The shell script used in this monitor is listed in Figure 46 on page 59.

```
#!/usr/bin/ksh
#####################################################################
#
# FILE:         checkdns
# VERSION:      1.00 04 February 1998
# AUTHOR:       Robi Banerjee, rob9@us.ibm.com
#
# (C) IBM Corp; 1998
#
#
# Background:
# The "nslookup" command is normally available on most of Unix systems
# and it does a name or IP resolution by querying the default nameserver.
# However, even if the nameserver is running and listening on the default
# port (53) , the only way to find out if it is resolving queries
# correctly is by sending it a query.
#
# The script will accept either an IP address or a fully qualified
# hostname and will check to see whether there was a match between
# the return value of the nslookup command. If a match is found between
# the userinput and the value returned by the nslookup command the
# script returns a "Success" message.
# Otherwise it will return a "Failure" message.
# This script can be used to show that the domain name server is
# accepting and resolving requests correctly. If a partial hostname
# is defined as input the "nslookup" command searches only the default
# domain name.
#
# Shell has been tested on AIX 4.2.0 .
#
#####################################################################
#
#Check to see that we have more than one argument
if [[ $# < 1 ]]; then
 echo "Usage:$0 [Internet Address]"
 exit 1
fi
queryval=$1
nslookup $queryval | awk -v param1=$queryval '
#compare first field of the nslookup output
$1 ~ /Name:/ {
#check to see if query value is part of return string from
nslookup
namepos=match($0,param1)
}
$1 ~ /Address:/ {
#check to see if query value is part of return string from
nslookup
addrpos=match($0,param1)
}END { #check to see if match was found either by Name or Address
if (namepos > 0 || addrpos > 0)
print "Success"
else
print "Failure"
}' 2> /dev/null
```

Figure 46. Check DNS Shell Script

The script first checks the correct number of arguments and then performs an nslookup. The output of this command is then checked to see if a match was found, that is the name server was able to resolve the IP address or hostname. Depending on the result, the word "Success" or "Failure" is returned to Tivoli Distributed Monitoring by printing it to stdout.

Note

The script should work on most operating systems where nslookup can be found in the system $PATH environment variable. However, if the default name server is not defined in the /etc/resolv.conf the return value from this script cannot be defined.

3.6 Query Well-Known Service Application Ports

In this section we create a monitor to check if an application is accepting connections on a TCP/IP port. This can be used to check if a client/server application is alive and responding to requests.

3.6.1 Monitor Overview

Just as it has become customary in a network environment to check whether we can reach a host using the ping command, it is also usual to check whether an application is listening on a known specified port. This confirms that the application or daemon is running. You can try to do this with the telnet command on most of the UNIX systems and you should get a login prompt. For example, you normally telnet to an IP address; however, if you specify a port number after the IP address the telnet client will try to connect to that specific port. The following example illustrates this:

```
root@rs600019:/tmp# telnet 9.24.104.60 9494
Trying...
Connected to 9.24.104.60.
Escape character is '^]'.
```

We tried connecting to port 9494 on the host with an IP address of 9.24.104.60 (rs60009). This host is also set up as an LCF gateway and hence we have a daemon active on port 9494. Break out of the telnet session by pressing a Ctrl+C. Enter `quit` to end the telnet session. However, this process of checking whether a port is active is impractical.

We can use the monitor defined in this example to check whether an application is active on a port, and based on the result we can notify an administrator if the application is down or not active. The well-known service applications have all their ports listed in the /etc/services file on a UNIX system.

This monitor is using Monitoring Capability Specification Language once more, however, with a slight difference. It references a C program in the monitor definition file and there are some differences that we point out later in this example.

3.6.2 Creating the Message Catalog

The message catalog file for this monitor is included below. We will not go into explaining the syntax of the message catalog file here as this was explained in the previous examples.

```
$ key=About_MonCol
1 This monitor attempts to connect to a port number specified
by the user on an Internet address. If it connects successfully
it returns a Connected message string otherwise it will return
an Error message string.
$ key=MonSrc_Descr1
2 Connect to Application Port
$ key=MonSrc_val_Descr1
3 (Result String)
$ key=About_MonSrc_Descr1
4 This monitor accepts two parameters as arguments, one of which
is an IP Address and the second one is an application Port number.
It will attempt to connect to the application port specified.
$ key=host_address
5 Host IPAddress
$ key=host_port
6 Application Port#
```

Figure 47. Portquery Message Catalog File (portquery.msg)

3.6.3 Compiling the Message Catalog

We compile the message catalog file (portquery.msg) in our example using the gencmsg command.

```
gencmsg portquery.msg
```

3.6.4 Creating the Monitor Definition File

The monitor definition file for this example is similar to the other monitor definitions that we have described in the previous examples. However, there are some differences worth noting.

Differences

- We invoke a C program binary and the Implementation section is different from invoking a shell script.
- The name of the program is passed in as the first parameter to the `Shell` keyword. Any parameters that are included after this program name will get passed to the program as arguments preceding the arguments that are passed in from the user through the `Argument` keyword.
- The monitor program defined will need to be found in the $PATH environment variable. You will otherwise get an E.EXEC error when your monitor is triggered through the TME 10 Distributed Monitoring 3.6 GUI.
- For our purposes we created a /usr/local/bin directory and added this to the $PATH script line in the setup_env.sh file. This file resides in the /etc/Tivoli directory and is normally executed to set up the Tivoli environment.
- It should be remembered that since this is a compiled binary it is not distributed to the managed node automatically. It is the administrator's responsibility to get the binaries to the managed node in the specified directory.

The monitor definition file for this example is shown in Figure 48.

```
#include "Sentry2_0.dsl"
#include "portquery.dsl"

Collection "CheckApplnPort" {
 CodeID = "$Id:portquery.csl,v 1.0$";
 Version = "1.0";
 Require = ">2.0.2";
 HelpMessage = (portquery_About_MonCol);
 EventBaseClass = "Sample_Sentry_Monitors";
 NoticeGroup = "Sentry";

#include "operators.csl"
#include "formats.csl"

 Monitor CheckApplnPort String Group string {
 Description = (portquery_MonSrc_Descr1);
 ValueDescription = (portquery_MonSrc_val_Descr1);
 HelpMessage = (portquery_About_MonSrc_Descr1);
 Argument (portquery_host_address)
  DefaultValue "";
 Argument (portquery_host_port)
  DefaultValue "";

 Implementation (aix3-r2, aix4-r1)
 Shell("appnport")
 ;
 };

}
```

Figure 48. Portquery Monitor Definition File (portquery.csl)

As seen in Figure 48 the monitor definition file passes in two arguments to the compiled C program called appnport. These arguments are the IP address of the host and the application port number.

3.6.5 Compiling the Monitor Definition File

Use the mcsl command to compile the monitor definition file seen in Figure 48 on page 62.

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

3.6.6 Installing the Monitoring Collection

Install the monitoring collection using:

```
mcsl -i -R portquery.col
```

3.6.7 Restarting the Object Dispatcher

You can now restart the object dispatchers on all the nodes so that you may start using this monitoring collection.

3.6.8 Using the New Monitoring Collection

Once the monitoring collection has been installed, you can use it by clicking on **Add Monitor...** in a normal Tivoli Distributed Monitoring profile. You will see the following window:

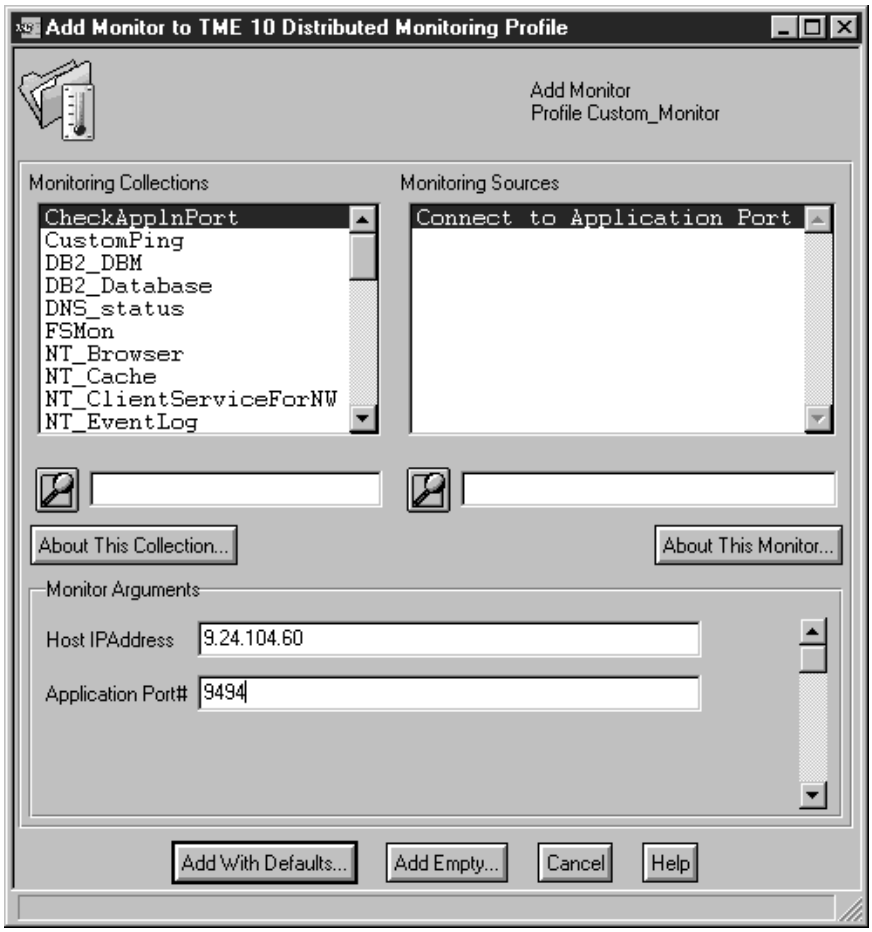


Figure 49. Add Monitor to TME 10 Distributed Monitoring Profile Window

Select the monitoring collection **CheckApplnPort** from the left-hand table and then click on **Connect to Application Port** in the right-hand table. Enter a Host IPAddress and an Application Port# in the list boxes displayed and click on **Add Empty...**.

In the Edit Monitor window that is displayed next, we select a response level of `Critical` when the program string Connected matches our input. We set up a pop-up to be displayed as an action, which can be seen as shown in Figure 50 on page 65:

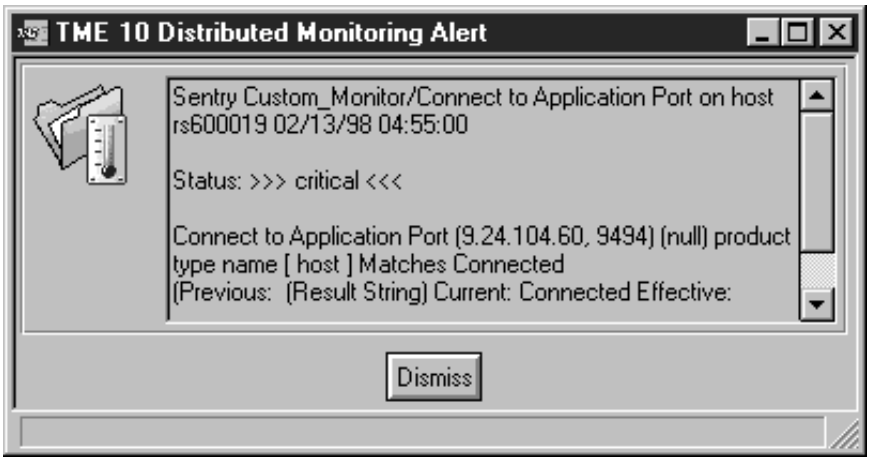


Figure 50. Alert from Application Port Query Monitor

In the above example, the machine 9.24.104.60 is listening to port 9494.

3.6.9 Source Code for the Port Query Monitor

The source code used for the monitor in this example is shown in the following figures.

```
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/errno.h>
#include <stdio.h>

extern int errno;

/*****************************************************************
*
* FILE:         userport.c
* VERSION:      1.00 02 February 1998
* AUTHOR:       Robi Banerjee, rob9@us.ibm.com
*
* COPYRIGHT:
*
*   All Rights Reserved
*   Licensed Materials - Property of IBM
*
*   The following code establishes a connection to an application
*   Port specified by the user to a given IP address. If the
*   connection is establised, the code generates a "Connect" message
*   signifying that the application is active. If no connection could
*   be established the code generates an "Error" message.
*   For checking whether the ftp daemon is active on a host you
*   can query the ftp port (21) using the following command :
*    appnport   9.24.104.249  21
*   For a list of well-known ports that can be queried please
*   refer to the /etc/services file on your host.
*
*****************************************************************/
int main(int argc, char **argv) {
int sock;    /* socket */
int  rc;    /* return code */
short  port;
struct sockaddr_in server;  /* socket structure */
char * ipaddr;
/* Check to see that the program has the correct number of arguments */
if ( argc < 3 ) {
 printf("SYNTAX: %s [IP_Address] [TCP_Port
]\n', argv[0]);
 exit(1);
}
port = atoi(argv[2xs]);
ipaddr = argv[1xs];
/* initialize the server structure */
server.sin_len = sizeof(&server);
server.sin_family = AF_INET;
server.sin_port = htons(port);
server.sin_addr.s_addr = inet_addr(ipaddr);
```

Figure 51. Application Port Monitor Source Code (Part 1 of 2)

```
/* Create socket: */
if((sock=socket(AF_INET, SOCK_STREAM, 0)) < 0) {
 printf("Error\n");
 rc = -1;
}
/* connect to the server */
if ( connect(sock, (struct sockaddr *) &server, sizeof(server))== -1) {
 printf("%s\n","Error");
 rc = -1;
}
else {
/* the connect was successful! */
 printf("%s\n","Connected");
 rc = 0;
}

close(sock);

exit(rc);

}
```

Figure 52. Application Port Monitor Source Code (Part 2 of 2)

The monitor first checks the correct number of arguments and then initializes a socket data structure. Once this is done, a connect to the server port is attempted. Depending on the success of this operation, 0 or -1 is returned.

The string "Error" or "Connected" is written to stdout to indicate the result to the Tivoli Distributed Monitoring monitor.

3.7 Using SNMP Query To Determine Router Statistics

This example demonstrates the use of a Tivoli Distributed Monitoring proxy to monitor devices not directly supported by TME 10 Distributed Monitoring 3.6.

The proxy feature is accomplished by running a monitor on a managed node that analyzes the availability of a host by checking the value of an SNMP MIB value on a non-TME managed node.

The response from the SNMP client is examined and a message is returned by the managed node running the monitor on behalf of the proxy endpoint defined in the TME 10 Distributed Monitoring 3.6 proxy.

3.7.1 Monitoring via SNMP

Describing the monitors available via SNMP is beyond the scope of this redbook. Refer to the redbook *IBM Systems Monitor Anatomy of a Smart Agent*, SG24-4398 for a detailed description of the SNMP monitors available.

The following SNMP collections are available in TME 10 Distributed Monitoring 3.6:

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

User-specified means that you can query any MIB-II variable that is supported by the SNMP agent on the non-managed TME host. This could be a proprietary MIB that the product supports, such as Lotus Notes, CascadeView, Cisco, Wellfleet, etc.

In our example, we specify a MIB-II variable named SysDescr on an IBM 2210 router. The same information can also be obtained by using the RFC 1213 SNMP monitor called Host Description.

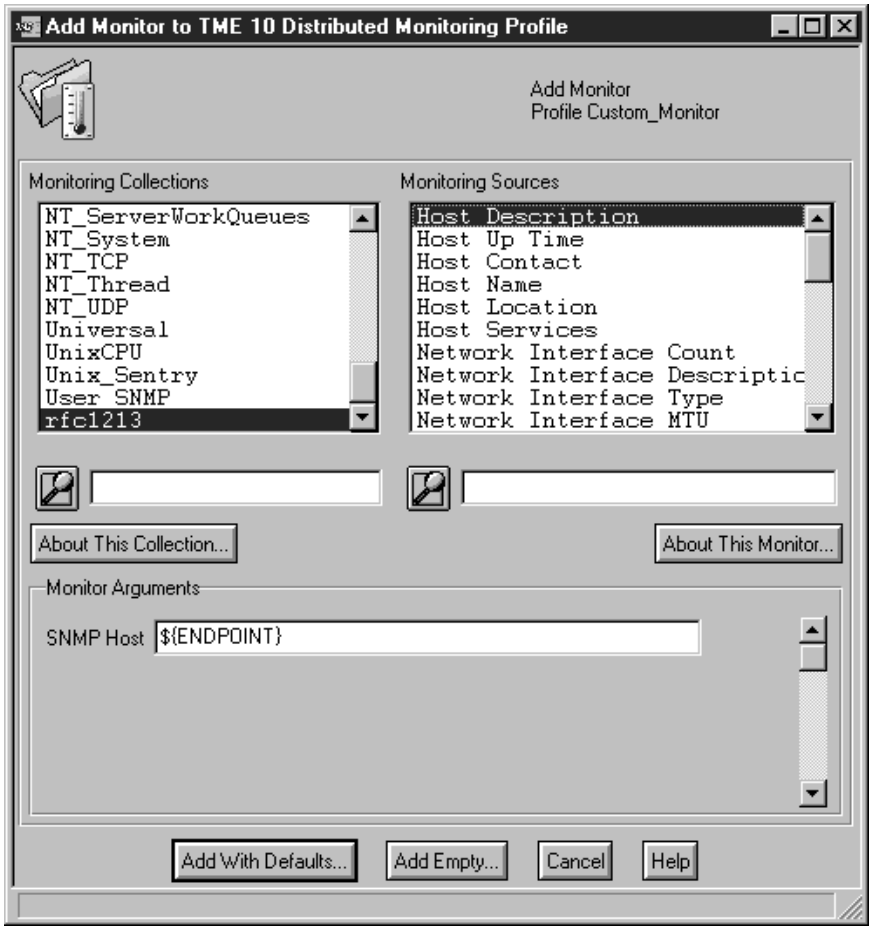


Figure 53. RFC1213 Monitor Using Symbolic Names for SNMP

However, the information returned using the RFC1213 monitor is a long string that is not practical to use in the trigger options provided by a String monitor. The result from such a monitor (Host Description) can be seen in Figure 54 on page 69.

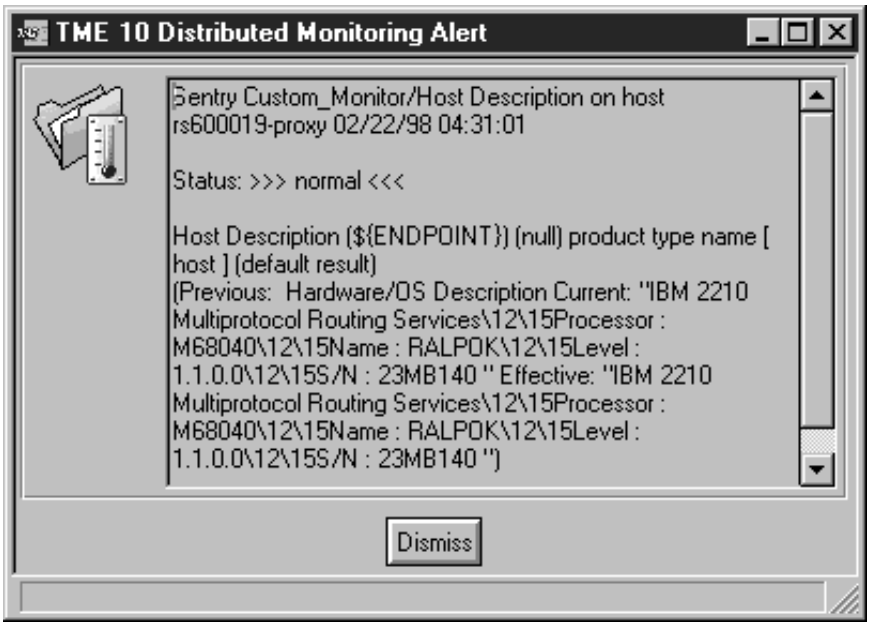


Figure 54. Result String from a Host Description RFC1213 Monitor

This is an example of how the same can be obtained using a user-specified MIB-II variable and how the script can be tailored so that it extracts only the information that you want. The output from the user-specified SNMP query is then a single string that makes using the trigger options for a String collection easier than shown in the example in Figure 54.

3.7.2 SNMP Monitor Example With a Proxy

We write our monitor in Perl and use wsnmpget to retrieve the value of the MIB-II variable. Perl is packaged with TME 10 Framework and wsnmpget is part of Tivoli Distributed Monitoring. No additional packages need to be installed.

The MIB-II variable that we use in our query has a symbolic name of SysDescr.0 and has an SNMP OID (object instance data) value of 1.3.6.2.1.1.1.0. The value returned from querying this MIB variable is a good indicator that the SNMP agent on the specified host is running and the node is up and responding. We use this in our example to query an IBM 2210 router and an IBM 6611 router.

We kept it simple but the same script could be modified so that it can read IP addresses and device types from a file, maintain a table internally with device types and IP addresses, and then (based on the response from the MIB) do a match to correspond device types with the ENDPOINT variable (such as an IP address) on the managed node.

Using this scenario it could monitor multiple non-TME resources from a single script on the same managed node.

The code for this script is included in Figure 55 on page 70.

```
#!/usr/local/Tivoli/bin/aix4-r1/contrib/bin/perl
#########################################################################
#
# FILE:         snmprout
# VERSION:      1.00 22 February 1998
# AUTHOR:       Robi Banerjee, rob9@us.ibm.com
#
#   (C) IBM Corp, 1998
#
# This is an example of a simple perl script that will use the
# wsnmpget command to query a SNMP enabled device. The MIB variable
# queried is SysDescr.0 whose SNMP oid is 1.3.1.6.1.2.1.1.1.0
# The query arguments are passed in as parameters to the script,
# these are:
#
# 1.${ENDPOINT} variable (this is how the IP address is passed in)
# 2.community string
# 3.MIB variable object ID
#
# The script is however, tailored to parse the response for a specific
# MIB-II variable called SysDescr.0. Based on the response from the
# snmp client it will print a Responded message or a Failed message.
#
#########################################################################
#set some variables here
$iodir = "/tmp";
$progname = $0;
#the first arg is the endpoint variable
$VAR_ENDPOINT = $ARGV[0];
#the second is the community string
$COMMUNITY_NAME = $ARGV[1];
#the third arg is the mib object id
$MIB_VAR = $ARGV[2];

#extract just the endpoint variable name from the first argument
#this is normally passed in as ${NAME}
$VAR_ENDPOINT =~ s/(^\$\{)//;
$VAR_ENDPOINT =~ s/(})//;


#get the value of the endpoint from the environment variable
$HOST_IPADDRESS = $ENV{$VAR_ENDPOINT};

#this is the command string to be executed
$CMD="wsnmpget -t 5 -c $COMMUNITY_NAME -h $HOST_IPADDRESS $MIB_VAR
";
```

Figure 55. SNMP Query of a MIB Variable Using Perl (Part 1 of 2)

```
#redirect stderr to a log, because when wsnmpget times out it writes
#to stderr , Sentry kills script with EXEC 1 error
$logfile = "$iodir/$HOST_IPADDRESS" . "\.log";
open(STDERR,">$logfile") || die "$progname: could not open
\"$logfile\"\n";
# unbuffer I/O:
select(STDERR);
$| = 1;
select(STDOUT);
$| = 1;
#this executes the command in perl and gives us a open
#file descriptor
open(CMD_CHAN,"$CMD |") || die("Failed\n");

while (<CMD_CHAN>) {
 chop;
 ( $fld1, $fld2, $rest) = split(/\s+/,$_,3);
 $DEVICE_NAME = $fld2;
 close(CMD_CHAN);
} #end of while

if ( $HOST_IPADDRESS eq "9.24.104.3" ) {
 if ( $DEVICE_NAME eq "2210" ) {
  print "Responded\n";
 }
 else {
  print "Failed\n";
 }
}
elsif ( $HOST_IPADDRESS eq "9.24.104.1" ) {
 if ( $DEVICE_NAME eq "6611" ) {
  print "Responded\n";
 }
 else {
  print "Failed\n";
 }
}
else {
 if ( $DEVICE_NAME eq "" ) {
  print ("Failed\n");
 }
#cleanup
if ( -e $logfile ) {
  rm $logfile 2>&1
}
}
```

Figure 56. SNMP Query of a MIB Variable Using Perl (Part 2 of 2)

We use MCSL in this example as well, to create a message catalog and a monitor definition file that invokes the Perl script shown in Figure 58 on page 73. The message catalog that is used in this monitoring collection is displayed in Figure 57 on page 72.

```
$ key=About_MonCol
1 This monitor uses the snmpinfo command to send a query to a snmp
enabled host.It will then analyze the response from the query and
will let the user know whether the host has responded to the snmp
query.
$ key=MonSrc_Descr1
2 Send SNMP Query
$ key=MonSrc_val_Descr1
3 (Response)
$ key=About_MonSrc_Descr1
4 This monitor uses the sysDescr.0 MIB-II variable to query the host
whose IP address is entered below. It will return a Responded
message or Failed message based on the response from the server.
$ key=query_value
5 Host IP Address:
$ key=community_name
6 SNMP Community Name
$ key=mib_value
7 SNMP MIB Entry
```

Figure 57. Message Catalog File (routproxy.msg)

We compile the monitor definition file as shown in Figure 58 on page 73 to produce the routproxy.col file that we then install on the TMR server using the mcsl command.

After installing the .col file you should see a new monitoring collection called User SNMP in your Add Monitor window as seen in Figure 62 on page 76. The monitor definition file used in the SNMP monitor is listed in Figure 58 on page 73.

```
#include "Sentry2_0.dsl"
#include "routproxy.dsl"

Collection "User SNMP" {
 CodeID = "$Id:routproxy.csl,v 1.0$";
 Version = "1.0";
 Require = ">2.0.2";
 HelpMessage = (routproxy_About_MonCol);
 EventBaseClass = "Sample_Sentry_Monitors";
 NoticeGroup = "Sentry";

#include "operators.csl"
#include "formats.csl"

 Monitor queryMIB String Group string {
 Description = (routproxy_MonSrc_Descr1);
 ValueDescription = (routproxy_MonSrc_val_Descr1);
 HelpMessage = (routproxy_About_MonSrc_Descr1);
 Argument (routproxy_query_value)
  DefaultValue "";
 Argument (routproxy_community_name)
  DefaultValue "";
 Argument (routproxy_mib_value)
  DefaultValue "1.3.6.1.2.1.1.1.0";

 Implementation ( aix3-r2, aix4-r1 )
 Shell("/bin/sh", "-c", Command, "routproxy")
 Import "snmprout"
 ;
 };

}
```

Figure 58. Monitor Definition File (routproxy.csl)

3.7.3 Using the User SNMP Monitoring Collection

To see what our Perl script gets back in response from the SNMP agent, type the following on the system where the Sentry proxy will reside:

```
wsnmpget -h POK2210A -t 5 -c public 1.3.6.1.2.1.1.1.0
```

In the above command, POK2210A is the hostname of the SNMP client, 5 is the timeout value, public is the SNMP community name and 1.3.6.1.2.1.1.1.0 is the SNMP object. In place of POK2210A we could have used an IP address as well as, we see later in our examples in Figure 62 on page 76, and Figure 61 on page 75.

The response from this command is shown in Figure 59.

```
"IBM 2210 Multiprotocol Routing Services\12\15Processor :
M68040\12\15Name : RALPOK\12\15Level : 1.1.0.0\12\15S/N : 23MB140 "
```

Figure 59. Output from wsnmpget Command

The Perl script parses the output and does a match on the type of router (IBM 2210, IBM 6611, etc.) based on the hostname. It will then output a `Responded` message string or a `Failed` message string back to the calling program (Tivoli Distributed Monitoring).

3.7.4 Setting Up a Sentry Proxy Endpoint

Every client that you want to run this monitor on will have to be defined in the Tivoli database. To achieve this we have to create a proxy endpoint. We do this by selecting **Create** from the menu bar of the rs600019-region policy region and then selecting **SentryProxy...** from the pull-down menu. The window shown in Figure 60 will appear.

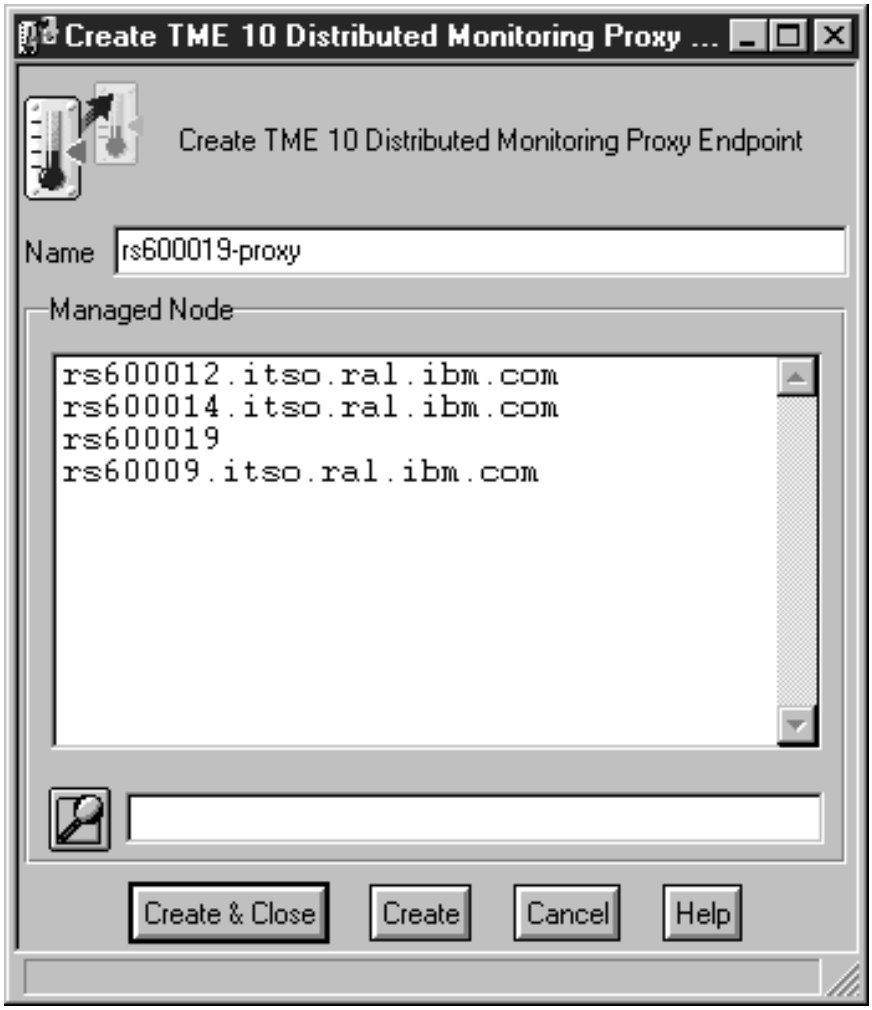


Figure 60. Create a TME 10 Distributed Monitoring Proxy Endpoint Window

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

This will create an icon in the policy region representing the new proxy endpoint. Double-clicking on the icon will open up the proxy endpoint window. Select **Configure** from the menu bar and then **Set Environment...** from the pull-down menu. The window shown in Figure 61 on page 75 will appear.

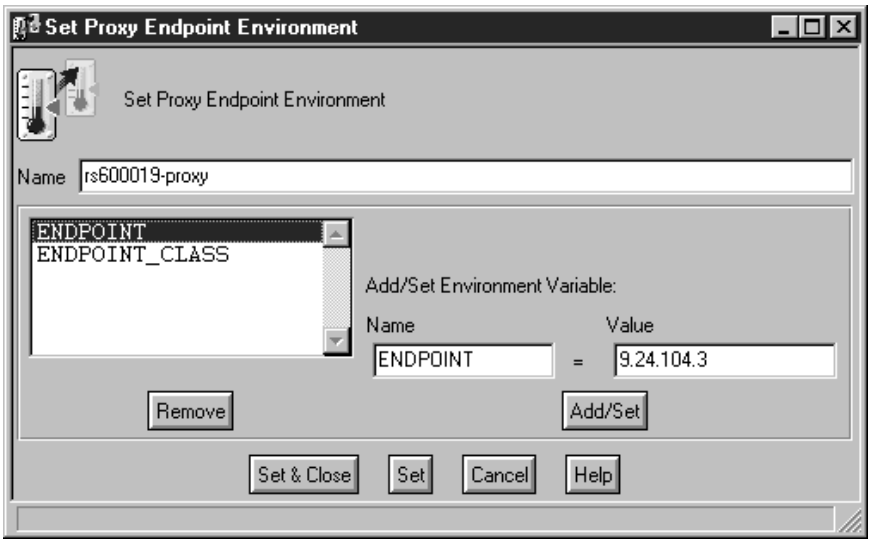


Figure 61. Set Proxy Endpoint Environment Window

We enter `ENDPOINT` in the Name field and the IP address of the router in the Value field and then click 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. Then we select the **Add/Set** button again.

The variable ENDPOINT contains the hostname of the client that we want to monitor and the variable ENDPOINT_CLASS is the Tivoli class name of the endpoint object. We will need to do this for every client that we need to monitor using the SNMP sysDescr.0 object.

Once we have set up a proxy endpoint and have configured the endpoint variables that we need to use, we are ready to use the SNMP proxy monitor. We need, however, to add the proxy endpoints created above as subscribers to the profile manager in which we will be adding our SNMP monitor profile.

We add our SNMP monitor to the profile in the usual manner. The Add Monitor to Tivoli Distributed Monitoring Profile window is displayed with our User SNMP collection. We pass in the parameters of the client that we need to monitor as shown in Figure 62 on page 76.

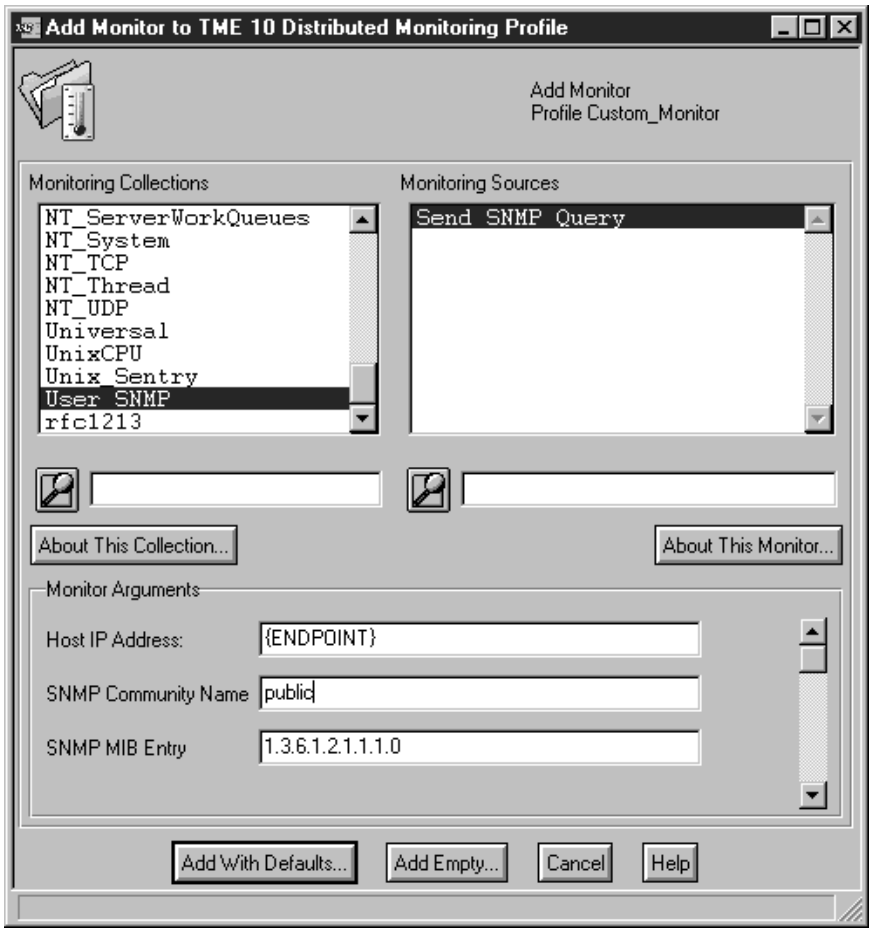


Figure 62. Add Monitor to TME 10 Distributed Monitoring Profile Window

The end result of this exercise is shown in Figure 63, which shows the "Responded" message when the monitor is set up with a pop-up action to Root@rs600019-region as an `Always` response.

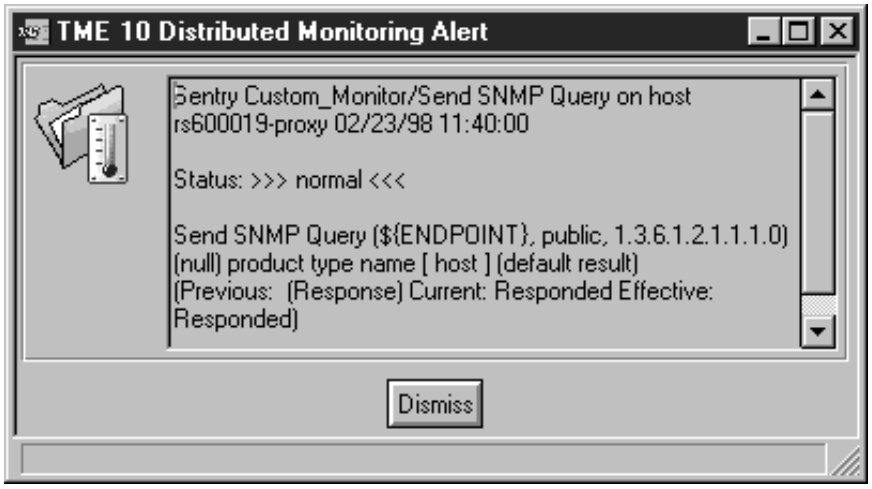


Figure 63. Response from the User SNMP Monitor

The SNMP proxy monitor can also be used with an indicator collection. In the next section, we detail the steps necessary for this purpose.

3.7.5 Creating an Indicator Collection

An indicator collection is added to a policy region by opening the policy region to which you want to add the indicator collection. In the policy region window select **Create** from the menu bar and then **Indicator Collection...** from the pull-down menu. Give the indicator collection a name (we use SNMP_Proxy_SysDescr) and select **Create & Close**.

This will create an icon in your policy region representing the new indicator collection. The next step is to add this indicator collection to the Tivoli Distributed Monitoring profile that will contain the monitors that you will use for testing out the different proxy endpoints. Select the Tivoli Distributed Monitoring profile from the policy region window and double-click it to open up the profile.

Select **Monitoring** from the menu bar and then select **Indicator Collection...** from the pull-down menu. Select the indicator collection that you have created above for monitoring the proxy endpoints from the list and select **Create & Close**.

In the next step as shown in Figure 64, you will add a monitor for the proxy using one of the endpoint variables that you defined in Figure 61 on page 75 (in our example we use ${ENDPOIN1}.) We set up a response level of `Critical` when we match the "Failed" output from the monitoring script. We also click on **Change Icon** to update our indicator collection that we associated with this Tivoli Distributed Monitoring profile.

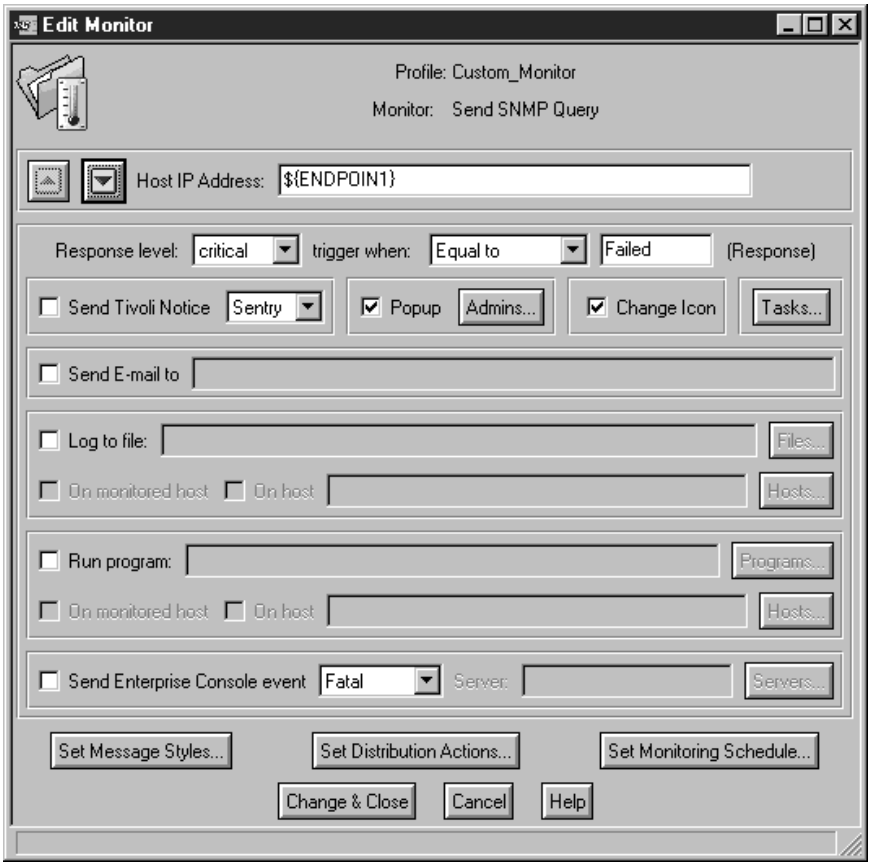


Figure 64. Adding a User SNMP Monitor With an Indicator Collection

Now distribute this profile, to the proxy endpoint that you set up in Figure 60 on page 74. Once your monitor is triggered, you can look at the indicator shown in Figure 65 on page 78 to see if there was something wrong with this SNMP variable on the proxy clients.

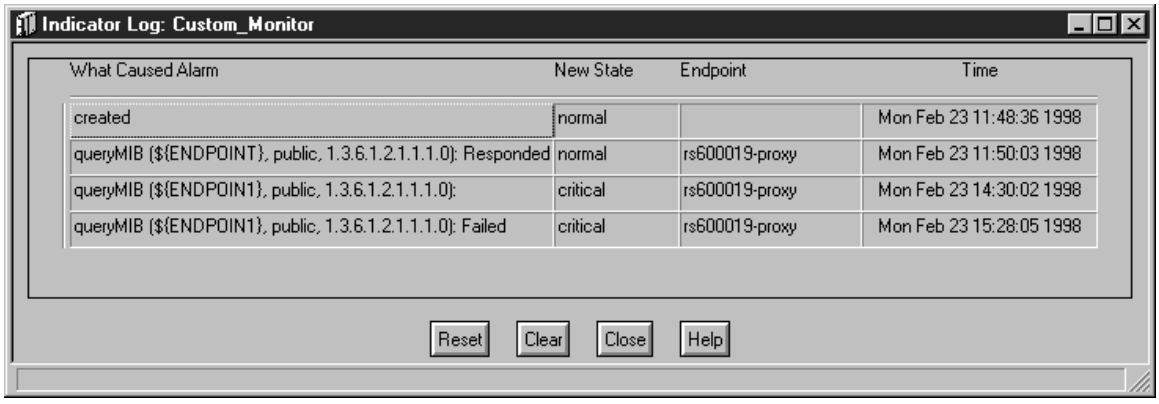


Figure 65. User SNMP Indicator Collection Messages

Note

The proxy clients are defined using endpoint variables that are unique for each client. We defined some endpoint variables in Figure 61 on page 75.

3.8 Generic Log File Parser

This generic log file parser monitor is an example of how a simple program may be written to monitor any generic log file for specific string patterns. The monitor can be applied to any ASCII log file. Log file entries are examined with the search criteria passed to the monitor. If the monitor finds log entries that match the criteria, it will send a numeric count of the number of such matching entries found. In our example we use it to monitor the syslog file for failed login attempts to a host that may signal to a system administrator that a host's security is being breached. We added the following entries to the /etc/syslog.conf file on AIX:

```
# example:
# "mail messages, at debug or higher, go to Log file.
#  File must exist."
# "all facilities, at debug and higher, go to console"
# "all facilities, at crit or higher, go to all users"
#  mail.debug           /usr/spool/mqueue/syslog
#  *.debug              /dev/console
#  *.crit                       *
*.debug         /var/adm/syslog
auth.debug      /var/adm/syslog
```

This will start logging all messages (including auth) to the /var/adm/syslog file specified. The syslog file has to exist before the messages start getting logged.

We use our log file parser to monitor the /var/adm/syslog file and pop up an alert, if the syslog file has entries that are specific for failed login attempts. The monitor will read the entire log the first time it is triggered and will examine only new entries

in the log, from the next time onwards. This monitor may be used in simple cases where a user wants to monitor an application log for specific messages.

3.8.1 Monitor Overview

The log file parser is a simple C program that opens an ASCII log file and reads each line as input. It will then parse this line and will try to match it against the search criteria that are entered by the user. The user-specified criteria are entered on a field basis, and this monitor is currently limited to searching two fields on a single line. The search entries can be logically AND'ed or OR'ed so that the user can specify a search criteria such as those that may be entered in a shell script.

Search_Criteria

```
$1 = oserv AND $3 = exiting
$4 = tsm:  OR $4 = klaxon
```

In the Add Monitor window where the search criteria are defined, the user would not have to enter the $ symbol; just the field number is required. Refer to the actual example in Figure 68 on page 83.

3.8.2 Creating the Message Catalog

The message catalog that we defined for the log file parser is shown in Figure 66 on page 80.

```
$ key=About_MonCol
1 This monitor is a generic logfile adapter. It will attach
to any logfile and monitor the logfile for specific field
values as specified by the user. It will return the number
of lines that matched the specified search criteria.
$ key=MonSrc_Descr1
2 Search Criteria
$ key=MonSrc_val_Descr1
3 (Occurences Found)
$ key=About_MonSrc_Descr1
4 This monitor will search the logfile that you specify with the
full pathname, and with the criteria defined in the parameters.
It will search the logfile each time from the last position read
for new data it found that match the search criteria.
$ key=ButtonLabelChoice
5 OpList
$ key=choice1_oper
6 AND
$ key=choice2_oper
7 OR
$ key=logfile_name
8 Logfile Name (full path)
$ key=field1_num
9 Enter (search) Field Number:
$ key=field1_val
10 Enter (search) Field Value:
$ key=field_oper
11 Operator:
$ key=field2_num
12 Enter (search) Field Number:
$ key=field2_val
13 Enter (search) Field Value:
```

Figure 66. Log File Parser Message Catalog (logfile.msg)

3.8.3 Compiling the Message Catalog

To compile the message catalog file (logfile.msg) in our example we use:

```
gencmsg logfile.msg
```

3.8.4 Creating the Monitor Definition File

The monitor definition file for this example is similar to the other monitor definitions that we have described previously. The monitor definition file (logfile.csl) can be seen in Figure 67 on page 81.

```
#include "Sentry2_0.dsl"
#include "logadap.dsl"

Collection "Custom_LogAdapter" {
 CodeID = "$Id:logadap.csl,v 1.0$";
 Version = "1.0";
 Require = ">2.0.2";
 HelpMessage = (logadap_About_MonCol);
 EventBaseClass = "Sample_Sentry_Monitors";
 NoticeGroup = "Sentry";

#include "choicelists.csl"
#include "operators.csl"
#include "formats.csl"

 ChoiceList DiffOptions {
  ButtonLabel = (logadap_ButtonLabelChoice);
  {
   { (logadap_choice1_oper) "AND" }
   { (logadap_choice2_oper) "OR" }
  };
 };

 Monitor GenLogAdapter Numeric Group numeric {
 Description = (logadap_MonSrc_Descr1);
 ValueDescription = (logadap_MonSrc_val_Descr1);
 HelpMessage = (logadap_About_MonSrc_Descr1);
 Argument (logadap_logfile_name)
  DefaultValue "";
 Argument (logadap_field1_num)
  DefaultValue "";
 Argument (logadap_field1_val)
   DefaultValue "";
 Argument(logadap_field_oper)
   RestrictedChoice "DiffOptions"
   DefaultValue "";
 Argument (logadap_field2_num)
  DefaultValue "";
 Argument (logadap_field2_val)
  DefaultValue "";

 Implementation (aix3-r2, aix4-r1)
 Shell("readlog")
 ;
 };

}
```

Figure 67. Log File Parser Definition File (logfile.csl)

Note

Since the monitor definition file also invokes a compiled C binary called readlog, be aware of certain requirements that were noted in our earlier example in Figure 48 on page 62. These apply for this monitoring example as well.

3.8.5 Compiling the Monitor Definition File

We use the mcsl command to compile the monitor definition file seen in Figure 67 on page 81.

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

3.8.6 Installing the Monitoring Collection

We install the monitoring collection using the mscl command:

```
mcsl -i -R logfile.col
```

3.8.7 Restarting the Object Dispatcher

You can now restart the object dispatchers on all the nodes so that you may start using this monitoring collection.

3.8.8 Using the New Monitoring Collection

As we stated in our introduction to this monitor, we use the monitor to examine the syslog file for failed login attempt messages. Using the Custom_Monitor profile manager that we set up in our environment, we select the **Add Monitor...** button. The following window is displayed, which should have the Custom_LogAdapter monitoring collection as shown:

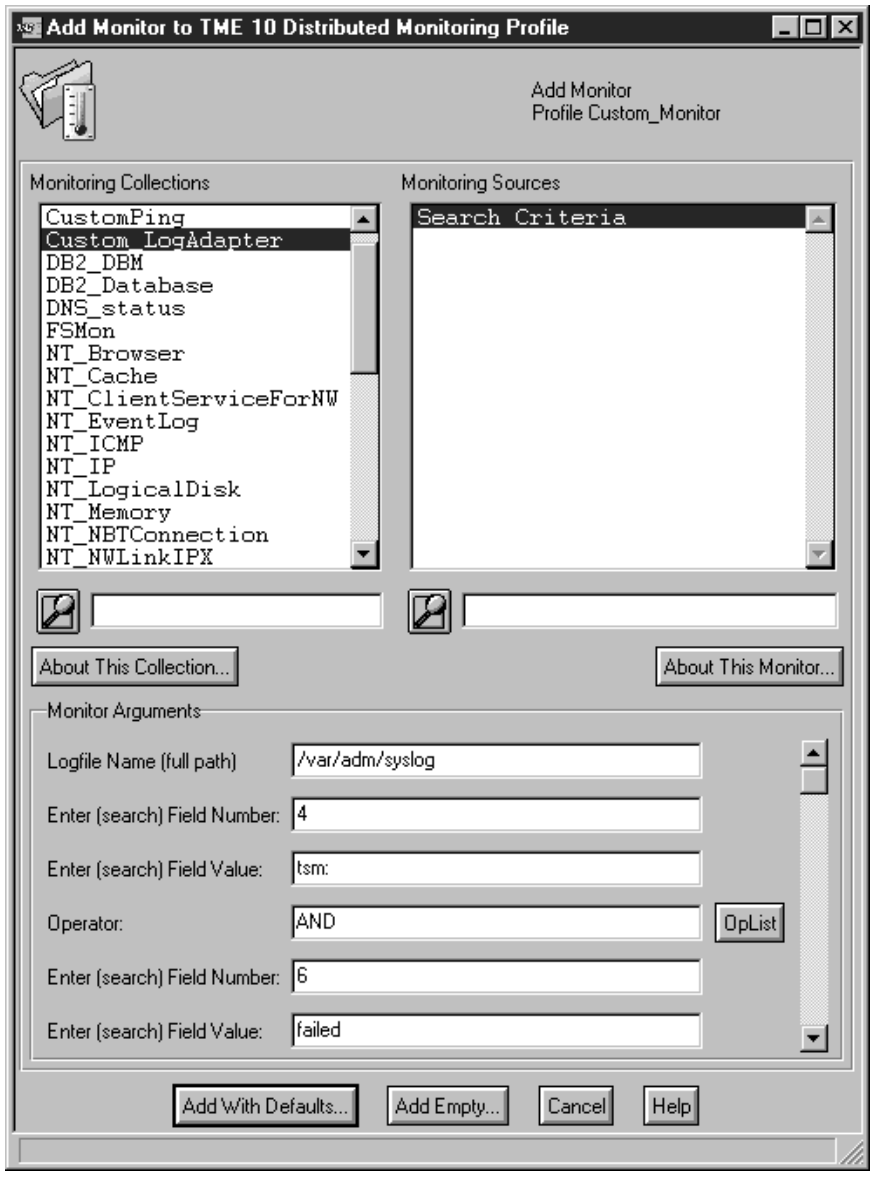


Figure 68. Add Monitor to TME 10 Distributed Monitoring Profile Window

Select **Custom_LogAdapter** from the Monitoring Collections section as shown and select **Search Criteria** from the Monitoring Sources section. Enter the search criteria as shown in Figure 68 and then select **Add Empty...**.

We set up a `Critical` response that is to be triggered when the expression is found more than once in the syslog file. We set an action of a pop-up window to be displayed to Root@rs600019-region. We then log on to the rs600019 host from a Windows NT host (wtr05099) and try to log in as root with an invalid password. We do this twice, which generates the `failed` messages in the syslog file. Refer to the Figure 69 on page 84 file to see the messages generated:

```
root@rs600019:/var/adm# tail syslog
Feb 18 08:17:32 rs600019 sendmail[19112]
: The alias database /etc/aliases.pag is out of date
Feb 18 08:17:32 rs600019 sendmail[19112]
: /etc/aliases:
There are 3 aliases. The longest is 9 bytes, with 46 bytes total.
Feb 18 08:47:32 rs600019 sendmail[19158]
: The alias database
/etc/aliases.pag is out of date
Feb 18 08:47:32 rs600019 sendmaillbracket.19158]
: /etc/aliases:
There are 3 aliases. The longest is 9 bytes, with 46 bytes total.
Feb 18 09:17:32 rs600019 sendmail[27184]
: The alias database
/etc/aliases.pag is out of date
Feb 18 09:17:32 rs600019 sendmail[27184]
: /etc/aliases:
There are 3 aliases. The longest is 9 bytes, with 46 bytes total.
Feb 18 09:24:50 rs600019 tsm: pts/3: failed login attempt for root
from WTR05099.itso.ral.ibm.com
Feb 18 09:24:50 rs600019 tsm: pts/3: failed login attempt for root
from WTR05099.itso.ral.ibm.com
Feb 18 09:24:53 rs600019 tsm: pts/3: failed login attempt for root
from WTR05099.itso.ral.ibm.com
Feb 18 09:24:53 rs600019 tsm: pts/3: failed login attempt for root
from WTR05099.itso.ral.ibm.com
root@rs600019:/var/adm#
```

Figure 69. Syslog File Entries Generated on Host rs600019

The end result of this exercise is the alert that is displayed on our screen, when the monitor examines the log file and matches the occurrences in the file with the search criteria we have specified. The alert is shown in Figure 70.

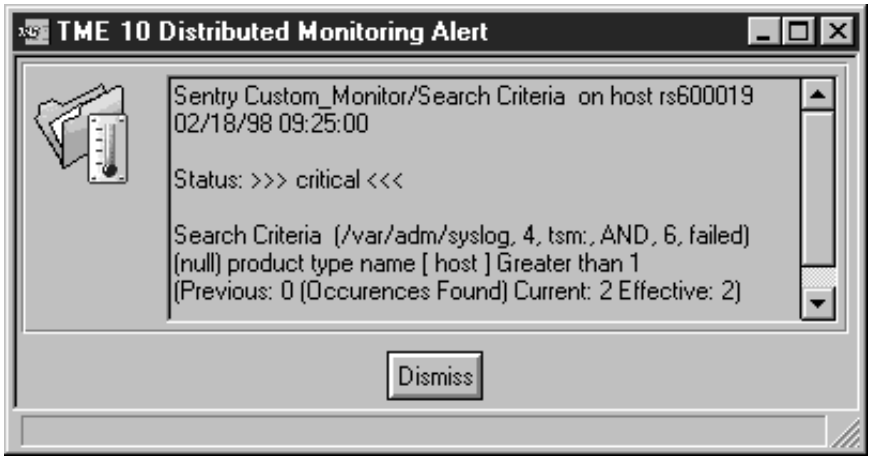


Figure 70. Alert Response from Log File Monitor

The following characteristics of the monitor need to be emphasized:

1. The log file being monitored needs to have its permissions set corectly so that the monitor can have read access to the log file during execution.

2. The monitor program readlog needs to be in the $PATH environment variable of the system.
3. The monitor program needs to have write permission to the directory in which the log file (being monitored) is located. This is because it creates a file with the same name as the log file but with a `.offset` extension for internal uses.
4. The monitor program will fail to work if the log file being monitored is rotated or truncated. You will have to delete the file with the `.offset` extension in case the log file is truncated or rotated.

3.8.9 Source Code for the Log File Parser

The source code for the log file parser used in this example is included below.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/errno.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/errno.h>

/* parse input string on newline,tab and space */
#define WHITESP "\n\t "
#define Boolean char
#define MAXBUF 4096
#define MAXFLDS 512

extern int errno;

/* save the user inputs in this structure */
struct SearchExpr {
 int field1;
 char * fldval1;
 int field2;
 char * fldval2;
 char * operator;
};

/* function prototypes */
int check_input(char ** ,struct SearchExpr * , int numflds );
int parse_input( char * , char **, char * );

/*********************************************************************
* FILE:         logfile.c
* VERSION:      1.00 17 February 1998
* AUTHOR:       Robi Banerjee, rob9@us.ibm.com
*
* (C) Copyright, IBM Corp. 1998
*
* Program Input: [logfile_to_monitor]
*   [Field1 # in String]
*   [Field1_Value in String]
*  [Operator (AND or OR)]
*  [Field2 # in String]
*  [Field2_Value in String]
*
* Program Output: int (Number of occurences found with above pattern)
*
* The program will open a logfile specified and will search each line
* in the log for specific patterns specfied with the input criteria.
* It will output the number of lines that the pattern was found in.
* This program calls two functions:
* parse_input => this function parses the input line into fields and
* returns the number of fields.
* check_input => this function returns a 1 if the line matched the input
* field criteria that was specfied by the user. Otherwsie it will return
* a zero.
```

Figure 71. Listing of logfile.c (Part 1 of 5)

```
* The program also saves the offset position of the logfile during the
* last read and will start reading the logfile from the last read
* position. The offset position is saved in a filename with the same
* name as the logfile with a ".offset" extension.
*********************************************************************/
int main(int argc, char ** argv) {

char * log_filename;
char * offsetfile;
FILE * rfp, * wfp;
int flds, num_found;
long offset;
char * FldVal[MAXFLDS];
char bufoffset[64];
char buffer[MAXBUF];
struct SearchExpr InputValues;

memset(&InputValues,0,sizeof(InputValues));

/* save the user input variables in the structure */
log_filename = argv[1];
InputValues.field1 = atoi(argv[2]);
InputValues.fldval1 = argv[3];
InputValues.operator = argv[4];
InputValues.field2 = atoi(argv[5]);
InputValues.fldval2 = argv[6];
num_found = 0;

/* create a file to save offset information so every iteration we */
/* do not read same old stuff */
if ( ( offsetfile = (char*) malloc(strlen(log_filename)+10) ) == NULL ) {
 printf(" malloc() error: errno returned %d\n", errno);
 exit(0);
}
else {
 strcpy(offsetfile,log_filename);
 strcat(offsetfile,".offset");
 /* check to see if the offset file exists */
 if ( ( wfp = fopen(offsetfile,"r") ) != NULL ) {
  fscanf(wfp,"%ld",&offset);
  fclose(wfp);
 }
 else {
  offset = 0;
 }
}

if ( ( rfp = fopen(log_filename,"r") ) == NULL ) {
 printf("error opening [%s] logfile\n", log_filename);
 exit(1);
}

if ( offset > 0 ) {
```

Figure 72. Listing of logfile.c (Part 2 of 5)

```
 fseek(rfp,offset,SEEK_SET);
}

while ( fgets(buffer,MAXBUF,rfp) != NULL ) {

 flds = parse_input(buffer, &FldVal[0], log_filename);
 if ( flds > 0 ) {
  num_found += check_input(&FldVal[0],&InputValues, flds);
 }

}

/* get the curent offset for the next read */
offset = ftell(rfp);
sprintf(bufoffset,"%ld\n",offset);
/* write the offset postion that we are reading to the offsetfile */
if ( ( wfp = fopen(offsetfile,"w") ) != NULL ) {
 fwrite(bufoffset,strlen(bufoffset),1,wfp);
}
/* cleanup and exit */
fclose(wfp);
fclose(rfp);
free(offsetfile);
printf("%d\n",num_found);
exit(0);

}


/*********************************************************************
* Function : check_input()
*
* Function Input: [ Input Array of fields ]
*     [ Structure that has the search expression
*                    specified by the user]
*     [ Number of fields in the string ]
*
* Function Output: 1 ( String with search pattern match )
*    0 ( String did not match search pattern )
*
*********************************************************************/
int check_input(char * FldVal[], struct SearchExpr * pinfo, int numflds ) {

int fld1, fld2;
Boolean found1, found2;

fld1 = pinfo->field1;
fld2 = pinfo->field2;
found1 = FALSE;
found2 = FALSE;

if ( (0 < fld1) && (fld1 <= numflds) ) {
 if ( strcmp(FldVal[fld1],pinfo->fldval1) == 0 )
```

Figure 73. Listing of logfile.c (Part 3 of 5)

```
  found1 = TRUE;
}

if ( (0 < fld2) && (fld2 <= numflds) ) {
 if ( strcmp(FldVal[fld2],pinfo->fldval2) == 0 )
  found2 = TRUE;
}

if ( strcmp(pinfo->operator,"AND") == 0 ) {
 if ( found1 && found2) {
  return 1;
 }
}
else if ( strcmp(pinfo->operator,"OR") == 0 ) {
 if ( found1 || found2) {
  return 1;
 }
}
/* user may not have selected an operator at all */
else if ( found1 == TRUE ) {
 return 1;
}

/* will be here only if both operators fail above */
return 0;

}

/*********************************************************************
* Function : parse_input()
*
* Function Input: [ Input String ]
*     [ Array of character pointers ]
*     [ logfile name ]
*
* Function Output: int ( number of fields in string )
*                  [ Filled array of charcater pointers ]
*
*********************************************************************/
int parse_input( char * buffer, char * FldVal[], char * fname ) {

int numflds;
char *ptr;

/* some minimal error checking */
if ( strlen(buffer) == 0 ) {
 return 0;
}

if ( strtok(buffer,WHITESP) == NULL ) {
 printf("Error parsing input in [%s] file\n", fname);
 return 0;
}
```

Figure 74. Listing of logfile.c (Part 4 of 5)

```
else {
 FldVal[0] = buffer;
 numflds = 0;
        while ( (ptr = strtok( NULL, WHITESP)) != NULL) {
                if (++numflds >= MAXFLDS) {
                        FldVal[--numflds] = NULL;
                        break;
                }
                else {
                        FldVal[numflds] = ptr;
                }
        }
        if (++numflds < MAXFLDS)
          FldVal[numflds] = NULL;
}

return(numflds);

}
```

Figure 75. Listing of logfile.c (Part 5 of 5)

3.9 Monitor to Download a Web Page

In this section we create a monitor that can be used to download an HTML file from a Web server and therefore test if a Web service is available.

3.9.1 Monitor Overview

We have extended the monitor defined in 3.6, “Query Well-Known Service Application Ports” on page 60 that connects to a user-specified port on a host machine. The monitor defined in this section connects to the httpd port (80) on a host and then issues a command to download a Web page specified by a URL path on this host.

We can use the monitor defined in this example to check a couple of things:

- The monitor lets us know that the Web service is alive on the host that we are connecting to.
- It will also let us know whether the Web service is responding to requests successfully.
- It will give a rough idea of the time it takes to download a home page from a remote location (where this monitor is activated).
- It will give the number of bytes read from the default home page. This should be a consistent figure every time the monitor is run. Any changes in byte count could mean that the Web page was changed recently.

3.9.2 Preparing to Run the Checkwebpage Monitor

This monitor is written using MCSL. The monitor definition file uses a Korn shell wrapper for a C program getweb, the source code for which is included in Figure 81 on page 97.

In preparation for running this monitor we suggest the following steps:

- Get the IP address of the host that you want to connect to.
- Confirm that there is an HTTP daemon running on this host (ask the administrator or use the ps -ef command on the host).
- Confirm that the HTTP daemon is listening on port 80. (Use the portquery monitor defined in 3.6, “Query Well-Known Service Application Ports” on page 60)
- Ask the Webmaster for a URL path that references an HTML document file. Ensure from the Webmaster that this document can be downloaded using a GET command and the access rights are set properly in the httpd configuration files.

  An example of a URL path is: `/project/nmsa/public_html/index.html/`
- Confirm that you can issue the command as defined in Figure 76 from the host that the monitor will run on.
- Confirm that there were no errors received from the command in the following figure. (You should see the HTML document downloaded or an error that the URL was not found, in which case you have to contact the Webmaster again for access rights and correct URL path.)

```
root@rs600019:/tmp# getweb [IP Address] [80] [URL path] [1]
```

Figure 76. Test Command before Running Webtime Monitor

The getweb monitor has a debug facility that can be used only from the command line. If you pass in a 1 as the fourth parameter, it will send the output to your display. This will let you know whether you can access the URL given by your Web administrator and whether the monitor will run from within the Tivoli environment.

When you know that you are downloading your HTML document successfully and are not getting any errors, you can install the monitor with the same parameters in TME 10 Distributed Monitoring 3.6. There is no easy debugging facility once your monitor is installed using MCSL. Hence, you should confirm that the monitor works from a command line before calling it using MCSL.

3.9.3 Creating the Message Catalog

The message catalog file for this monitor is included below. We will not explain the syntax of the message catalog file here, as this was done in the previous examples.

```
$ key=About_MonCol
1 This monitor connects to the HTTPD port (80) on an Internet address
specified. It then attempts to download a WEB page specified
by the URL path on the given host.
$ key=MonSrc_Descr1
2 Download WEB page
$ key=MonSrc_val_Descr1
3 (Result)
$ key=About_MonSrc_Descr1
4 This monitor accepts four arguments; first is an IP Address, second
argument defaults to 80 which is the httpd port, third is a full path
to the URL on a host and fourth is the output choice that should be
returned from the monitor. It can return the total bytes downloaded
(bytes) or the time in seconds to download the WEB page (time).
$ key=host_address
5 Host IPAddress
$ key=host_port
6 Application Port:
$ key=host_url
7 Full Path to URL:
$ key=ButtonLabelChoice
8 Select:
$ key=choice_time
9 time
$ key=choice_bytes
10 bytes
$ key=choice_options
11 Options:
```

Figure 77. Message Catalog File (getweb.msg)

3.9.4 Compiling the Message Catalog

We compile the message catalog file (getweb.msg) in our example using:

```
gencmsg getweb.msg
```

3.9.5 Creating the Monitor Definition File

The monitor definition file for this example is similar to the other monitor definitions that we have described in the previous examples.

The monitor definition file for this example is shown in Figure 78 on page 93.

```
#include "Sentry2_0.dsl"
#include "webtime.dsl"

Collection "CheckWebPage" {
 CodeID = "$Id:webtime.csl,v 1.0$";
 Version = "1.0";
 Require = ">2.0.2";
 HelpMessage = (webtime_About_MonCol);
 EventBaseClass = "Sample_Sentry_Monitors";
 NoticeGroup = "Sentry";

#include "choicelists.csl"
#include "operators.csl"
#include "formats.csl"

 ChoiceList DiffOptions {
  ButtonLabel = (webtime_ButtonLabelChoice);
  {
   { (webtime_choice_time) "time" }
   { (webtime_choice_bytes) "bytes" }
  };
 };

 Monitor CheckWebPage Numeric Group numeric {
 Description = (webtime_MonSrc_Descr1);
 ValueDescription = (webtime_MonSrc_val_Descr1);
 HelpMessage = (webtime_About_MonSrc_Descr1);
 Argument (webtime_host_address)
  DefaultValue "";
 Argument (webtime_host_port)
  DefaultValue "80";
 Argument (webtime_host_url)
  DefaultValue "";
 Argument(webtime_choice_options)
   RestrictedChoice "DiffOptions"
   DefaultValue "";

 Implementation (aix3-r2, aix4-r1)
 Shell("webtime")
 ;
 };

}
```

Figure 78. Getweb Monitor Definition File (getweb.csl)

As seen in Figure 78 the monitor definition file passes in four arguments to the Korn shell wrapper called webtime.

These arguments are:

1. IP address of the host
2. Application port (80)
3. Full path to the URL that is to be downloaded
4. User choice of output needed (bytes|time)

The shell script calls a C program getweb that does the real work.

Note

For reasons not explained we found that the C program did not work when we called it directly from the monitor definition file. It gave an E.EXEC error and exited with an error code of 46. The same monitor works when we put the Korn shell wrapper around it as can be seen in Figure 80 on page 96.

3.9.6 Compiling the Monitor Definition File

We use the mcsl command to compile this monitor definition file:

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

3.9.7 Installing the Monitoring Collection

We install the monitoring collection using:

```
mcsl -i -R getweb.col
```

3.9.8 Restarting the Object Dispatcher

You can now restart the object dispatchers on all the nodes so that you may start using this monitoring collection.

3.9.9 Using the New Monitoring Collection

Once the monitoring collection has been installed, you can use it by clicking on **Add Monitor...** in a normal Tivoli Distributed Monitoring profile. You will see the following window:

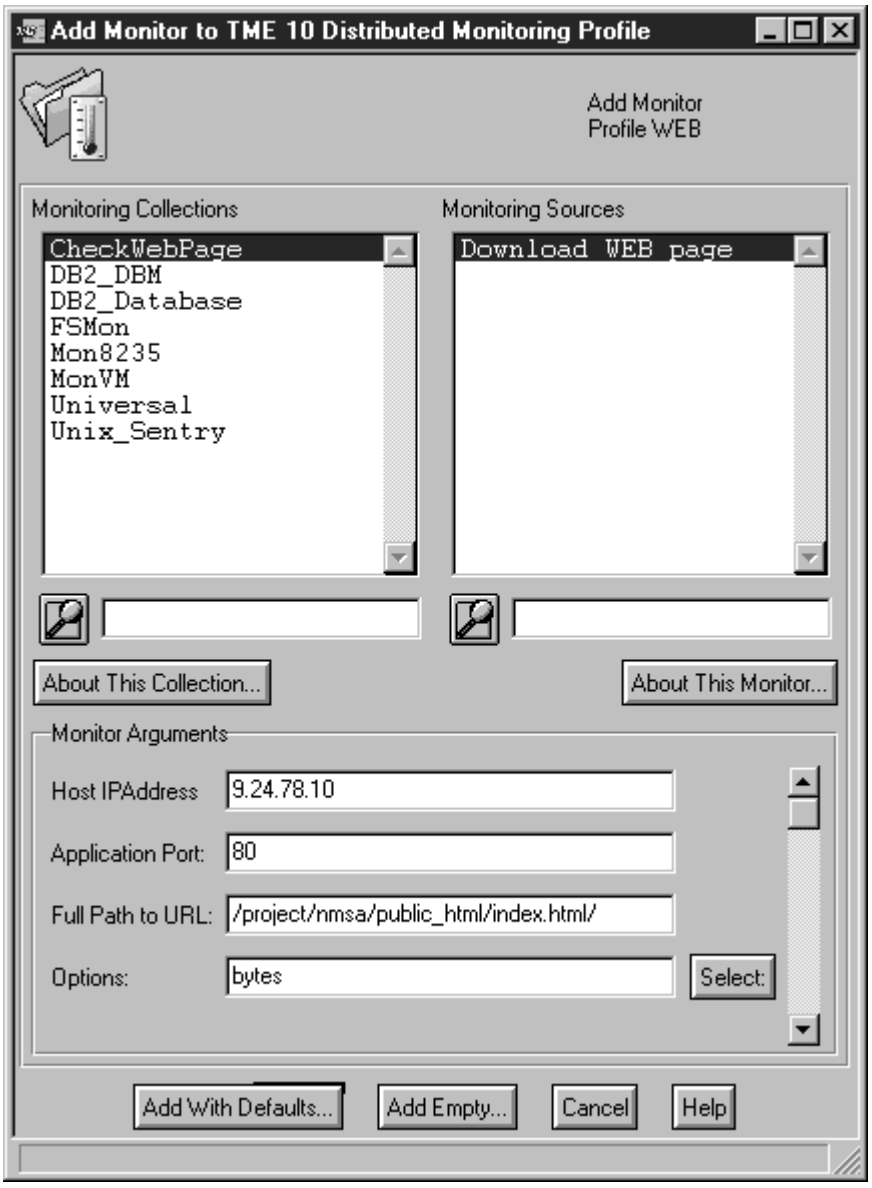


Figure 79. Add Monitor to TME 10 Distributed Monitoring Profile Window

Select the monitoring collection **CheckWebPage** from the left hand-table and then click on **Download WEB Page** in the right-hand table. Enter the information in the list boxes in the Monitor Arguments section. In our example, we download a default Web page from a URL that we have access to.

We enter the IP address of the host on which this URL exists as the first argument.

The second argument is left unchanged since we are connecting to port 80, which is the default httpd port.

The next list box specifes the fully qualified URL. The URL specified here needs to be granted the proper access rights (see your Webmaster) so a direct access is possible to this file. (Look at Figure 76 on page 91.)

The next list box specifies the options, `time` or `bytes`. Based on the option selected, the program will either output the total bytes read downloading the page, or the time needed to download the page.

In the Edit Monitor window that is displayed next, we select a response level of `Critical` when the program output is 0 (for bytes downloaded). We set up a pop-up to be displayed as an action, which can be seen as shown in Figure 80:

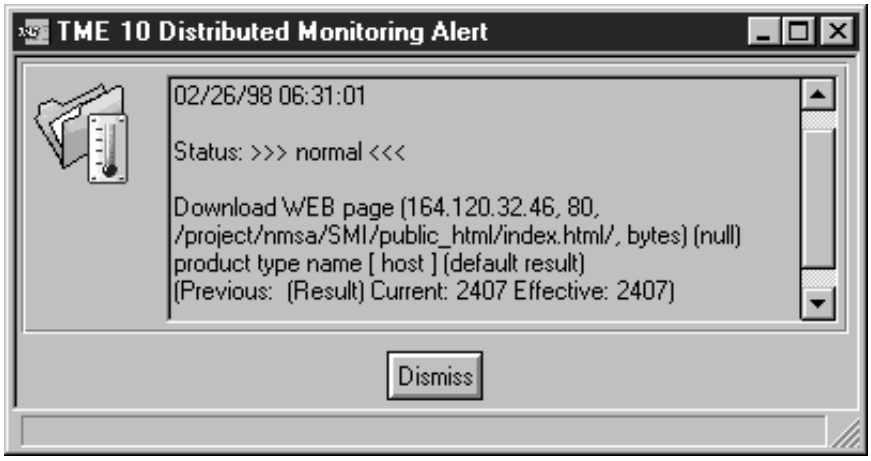


Figure 80. Alert from CheckWebPage Monitor

In the above example, the response is normal, as 2407 bytes could be downloaded from this Web page.

3.9.10 Source Code for the CheckWebPage Monitor

The source code used for the monitor in this example is included in the following figure.

```
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <string.h>
#include <time.h>
#include <sys/errno.h>
#include <stdio.h>

#define Boolean char
#define READBUF 1024
extern int errno;

/*********************************************************************
* FILE:         webtime.c
* VERSION:      1.00 24 February 1998
* AUTHOR:       Robi Banerjee, rob9@us.ibm.com
*
* COPYRIGHT:
*
*   All Rights Reserved
*   Licensed Materials - Property of IBM
*
*   The following code establishes a connection to an application Port
*   specified by the user to a given IP address. If the connection is
*   estabilised,
*
*********************************************************************/
int main(int argc, char **argv) {

char rbuf[READBUF];
int sock;    /* socket */
int  rc, bytesread, cmd_len, total;
short  port = 0;
struct sockaddr_in server;  /* socket structure */
char *ipaddr, * url , * cptr;
time_t start_time, end_time;
Boolean show;

/* Check to see that the program has the correct number of arguments */
if ( argc < 4 ) {
printf("SYNTAX: %s [IP_Address] [TCP_Port] [URL Path] [choice] \n",
argv[0]);
exit(1);
}

ipaddr = argv[1];
if ( ipaddr == "" ) exit(1);
port = atoi(argv[2]);
if ( port == 0 ) exit(1);
url = argv[3];
if ( url != NULL ) {
```

Figure 81. Listing of getweb.c (Part 1 of 3)

```
 cptr = strchr(url,'\n' );
 if ( cptr != NULL ) {
  *cptr = '\0';
 }
}
else {
 exit(1);
}
if ( argv[4] != NULL ) {
 cptr = strchr(argv[4],'\n' );
 if ( cptr != NULL ) {
  *cptr = '\0';
 }
}
else {
 exit(1);
}

/* only to be used for debug from command line */
if ( strcmp(argv[4],"1") == 0 )
 show = TRUE;
else
 show = FALSE;

/* initialize the server structure */
server.sin_len = sizeof(&server);
server.sin_family = AF_INET;
server.sin_port = htons(port);
server.sin_addr.s_addr = inet_addr(ipaddr);


/* Create socket: */
if((sock=socket(AF_INET, SOCK_STREAM, 0)) < 0) {
 printf("Error\n");
 rc = -1;
}
/* connect to the server */
if ( connect(sock, (struct sockaddr *) &server, sizeof(server)) == -1) {
 printf("%s\n","Error");
 rc = -1;
}
else {
/* the connect was successful! */
 memset(rbuf,0,READBUF);
 strcpy(rbuf,"GET ");
 strcat(rbuf,url);
 strcat(rbuf,"\n");
 cmd_len = strlen(rbuf);
 time(&start_time);
 if (( rc = write(sock, rbuf, cmd_len)) == -1) {
  printf("%s: unable to write to socket\n", argv[0]);
  close(sock);
  exit(1);
```

Figure 82. Listing of getweb.c (Part 2 of 3)

```
 }
}
/* if we encountered any error above no point going further */
if ( rc == -1 ) {
 exit(1);
}

total = 0;
while ((bytesread = read(sock, &rbuf[0], READBUF)) > 0) {
 total += bytesread;
 if ( show ) {
  write(1,rbuf,bytesread);
 }
 memset(rbuf,0,READBUF);
}
time(&end_time);
close(sock);
if ( strcmp(argv[4],"time") == 0 ) {
 printf("%d\n",end_time-start_time);
}
else {
 if ( strcmp(argv[4],"bytes") == 0) {
  printf("%d\n",total);
 }
 else {
  if ( show ) {
   printf("bytes read: [%d] time: [%d] seconds\n", total,end_time-start_time);
  }
 }
}
exit(rc);

}
```

Figure 83. Listing of getweb.c (Part 3 of 3)

The program first initializes a socket data structure and then opens a socket to connect to the specified HTTP server. If the connection is successful, a GET command is sent to the HTTP server in order to retrieve the specified URL.

Before and after reading the page, the time is taken, so the difference can be used to calculate the time needed to download the page.

3.10 Monitoring an IBM 8235

The IBM 8235 is a widely used device for remotely connecting to a LAN. It provides transparent LAN access to users dialing in using a modem connection.

The IBM 8235 is usually administered from a PC, in our case a machine running Windows 3.1. As it is difficult implementing the monitoring code directly on Windows 3.1, we will employ a different approach in this example.

This approach can be used for any network device that provides a telnet login to perform management operations.

The following figure illustrates the architecture of our monitor:

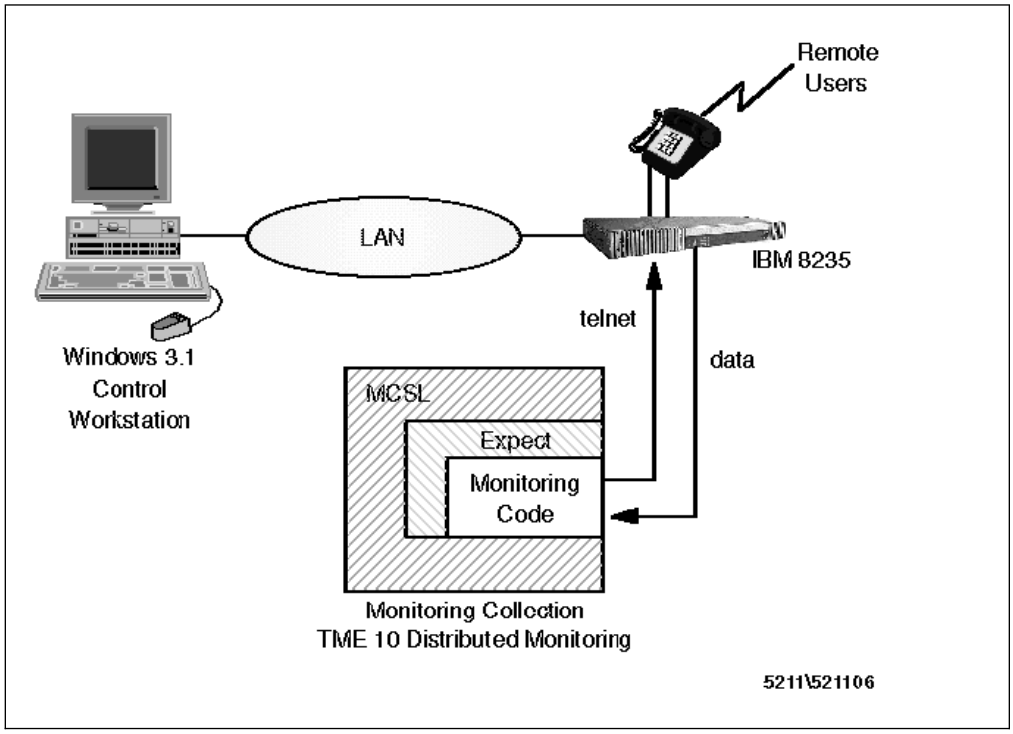


Figure 84. Monitoring an IBM 8235

There are two ways to extract data from the IBM 8235. One is using SNMP, but the MIB provided with the IBM 8235 has only very limited information. The other way is to use telnet. By using telnet you can access lots of information by using the show command that is available on the IBM 8235 after you telnet into it.

We will only look at a subset of this information, namely buffers and IP traffic.

Let's first have a closer look at the architecture of our monitor. As telnet is an interactive command, we cannot use it directly within a monitor. To solve this problem we use a common UNIX tool called Expect that can be used to drive interactive applications in batch mode.

Expect provides similar commands to a Korn shell and we can therefore implement our monitoring code within this environment. The complete monitor is then wrapped into MCSL code to make it available as a monitoring collection in Tivoli Distributed Monitoring.

To be able to access the IBM 8235 through telnet, you need a user ID and password.

Let's have a look at the output of the show buffers command when executed interactively from within a telnet session to the IBM 8235:

```
IBM8235_32C00A> show buffers
158 buffers available (0 drops)
FREE       154     0 max         0 reqs         0 drops
ARP          0     1 max     72966 reqs         0 drops
DATA         2   155 max 311855363 reqs         0 drops
SERIAL       1    15 max  10210979 reqs         0 drops
TOKEN        1   220 max 384872245 reqs         0 drops

Tiny buffers (144 bytes)
    32 allocated 32 max
    32 free 19 min (max allowed: 32)
    622629 requested, 0 drops, 0 creates, 0 trims

Small buffers (604 bytes)
    8 allocated 8 max
    8 free 7 min (max allowed: 8)
    29 requested, 0 drops, 0 creates, 0 trims

Medium buffers (1532 bytes)
    48 allocated 142 max
    46 free 0 min (max allowed: 48)
    382117931 requested, 0 drops, 60457 creates, 60457 trims

Large buffers (4544 bytes)
    70 allocated 155 max
    68 free 0 min (max allowed: 70)
    324272071 requested, 0 drops, 5139 creates, 5139 trims
```

Figure 85. Output from show buffers Command

A similar output can be obtained using the command:

```
show IP traffic
```

```
IP:
  26358374 total datagrams received
  3110 datagrams with header errors
  12634828 datagrams forwarded
  3972982 datagrams with unknown protocol
  9017592 datagrams delivered
  609844 output datagram requests
  1257 datagrams fragmented
  2514 fragments created
UDP:
  875005 total datagrams received
  7702295 datagrams to invalid port
  6 datagrams dropped due to errors
  195236 output datagram requests
TCP:
  76 passive opens
  4 resets of established connections
  2218 segments received
  1784 segments sent
ICMP:
  438068 total messages received
  4275 messages dropped due to errors
  412806 output message requests
```

Figure 86. Output from show IP traffic Command

As with the other monitors in this redbook, our monitor needs to process this output and transform it into a format that can be processed by Tivoli Distributed Monitoring.

The challenge with this monitoring collection is to do a telnet to the IBM 8235 from a program and run the telnet session interactively. We use the Expect tool, which is available as freeware on the Internet. For AIX the Expect program and the prerequisites can be found on http://aixpdslib.seas.ucla.edu. Expect is written in the Tcl language and we use the following packages:

- Expect 5.22
- Tcl 7.6p2

  The Tcl language.
- Tk 4.2p2

  The toolkit for Tcl.

We choose to download the executables for AIX 4.1 so we just need to un-tar the files and can start playing.

The IBM 8235 monitoring collection consists of the following elements:

- mon8235.msg

  The message file, which is listed in Figure 87 on page 104.
- mon8235.csl

  The csl file, which is listed in Figure 88 on page 105.
- mon8235

The monitor itself is written completely in Expect, listed in Figure 89 on page 106.

3.10.1 IBM 8235 Monitor Definitions

```
$ key=mon82351_Descr
1 8235 Statistics
$ key=mon8235_Help
2 This Monitor will extract counters from the 8235 box.
This is done by telnet to the box and issuing show statistics command.
$ key=mon82351_val_Descr
3 Error counters
$ key=mon8235_node
4 Node
$ key=mon8235_command
5 Error counters
$ key=mon8235_ip
6 IP traffic errors
$ key=mon8235_udp
7 UDP errors
$ key=mon8235_icmp
8 ICMP errors
$ key=ButtonLabelChoice
9 Counters
$ key=mon82352_val_Descr
10 Free buffers
$ key=mon82352_Descr
11 8235 Buffers
$ key=generic_help
12 The monitor will extract information from an 8235.
12 For more information please use the Redbook SG24-5211,
Creating Custom Monitoring Collections.
```

Figure 87. Listing of mon8235.msg

The message file shown in Figure 87 defines the fields for the two 8235 monitors, `8235 Statistics` and `8235 Buffers`.

The csl file shown in Figure 88 on page 105 defines the arguments for the two monitors.

Note

Both the Expect program and the mon8235 program must be available in the $PATH on the platform where the monitor will run. We tried to have Tivoli Distributed Monitoring distribute the program mon8235 as an imported shell, but that failed.

```
#include "Sentry2_0.dsl"

#include "mon8235.dsl"

Collection "Mon8235" {
    CodeID = "$Id:mon8235.csl,v 1.0$";
    Version = "1.0";
    Require = ">2.0.2";
    HelpMessage = (mon8235_generic_help);
    EventBaseClass = "Sample_Sentry_Monitors";
    NoticeGroup = "Sentry";

#include "operators.csl"
#include "formats.csl"
#include "choicelists.csl"

    ChoiceList Counters {
      ButtonLabel = (mon8235_ButtonLabelChoice);
      {
         { (mon8235_mon8235_ip) "iptraffic" }
         { (mon8235_mon8235_udp) "udptraffic" }
         { (mon8235_mon8235_icmp) "icmptraffic" }
      };
    };

    Monitor Counters Numeric Group numeric{
      Description = (mon8235_mon82351_Descr);
      ValueDescription = (mon8235_mon82351_val_Descr);
      HelpMessage = (mon8235_mon8235_Help);
      Argument (mon8235_mon8235_node)
        DefaultValue "";
      Argument (mon8235_mon8235_command)
        RestrictedChoice "Counters"
        DefaultValue "iptraffic";
      Implementation (aix4-r1)
      Shell("mon8235");
  };
     Monitor Buffers Numeric Group numeric{
      Description = (mon8235_mon82352_Descr);
      ValueDescription = (mon8235_mon82352_val_Descr);
      HelpMessage = (mon8235_mon8235_Help);
      Argument (mon8235_mon8235_node)
        DefaultValue "";
      Argument (mon8235_mon8235_command)
        Restriction "buffers"
        DefaultValue "buffers";
      Implementation (aix4-r1)
      Shell("mon8235");
  };
}
```

Figure 88. Listing of mon8235.csl

```
#!/usr/local/bin/expect --

if {[llength $argv] < 2} {
      puts "Usage is mon8235 address command"
      exit 1
     }
set node [lindex $argv 0]
set cmd  [lindex $argv 1]
set env(TERM) vt100
set user ""
set file "/etc/security/mon8235.users"

# Disable stdout output
log_user 0

# Find out userid and password for the 8235
set fp [open $file r]

while {[gets $fp line] != -1} {
        scan $line "%s %s %s" wnode wuser wpass
        if {![string compare $wnode $node]} {
          set user $wuser
          set pass $wpass
          }
        }

close $fp

if {$user == ""} {
      puts "No matching nodeid found in mon8235.users file"
      exit 1
     }
# Start telnet session
spawn telnet $node
# Uncomment line below to get debug information to stderr
#exp_internal 1

expect "Userid: "
send "$user\r"
expect "Password? "
send "$pass\r"

expect "> "

if {$cmd == "buffers"} {
      send "show $cmd\r"
      expect -re "(\[^\r]* drops)\r\n"
      set 8235val [lindex $expect_out(1,string) 1]
      }

if {$cmd == "iptraffic"} {
      send "show ip traffic\r"
      expect -re "(\[^\r]* datagrams with header errors)\r\n"
      set 8235val [lindex $expect_out(1,string) 0]
      }
```

Figure 89. Listing of mon8235 (Part 1 of 2)

```
if {$cmd == "udptraffic"} {
       send "show ip traffic\r"
       expect -re "(\[^\r]* datagrams dropped due to errors)\r\n"
       set 8235val [lindex $expect_out(1,string) 0]
       }

if {$cmd == "icmptraffic"} {
       send "show ip traffic\r"
       expect -re "(\[^\r]* messages dropped due to errors)\r\n"
       set 8235val [lindex $expect_out(1,string) 0]
       }

# Logout from telnet
send "quit\r"

# Enable stdout output
log_user 1

# Return extracted value
send_user "$8235val"

exit
```

Figure 90. Listing of mon8235 (Part 2 of 2)

Note

You must set the TERM environment variable as the telnet process will run in the background. If it is not set, the telnet process will fail.

The file mon8235 shown in Figure 89 on page 106 starts with verifying the correct number of arguments being passed. Then stdout is disabled as we don't want the telnet session output echoed.

If we pass the user ID and password as arguments to the mon8235 Expect program, those arguments will be shown in the pop-up, which is a bad implementation as anybody can see the password. So, we need a way to avoid showing user ID and password in the pop-up. Therefore, we decide to have a file with node, user ID, and password.

We can then keep this file in a secure location, such as /etc/security, so that only the root user can access it.

The line `#exp_internal 1` can be uncommented to get debugging information to stderr. This information can be very useful when you have problems matching strings from output.

```
9.24.104.001 usera passa
9.24.14.132 userb passb
9.24.104.222 userc passc
```

Figure 91. Listing of mon8235.users

3.10.2 Running the 8235 Monitor

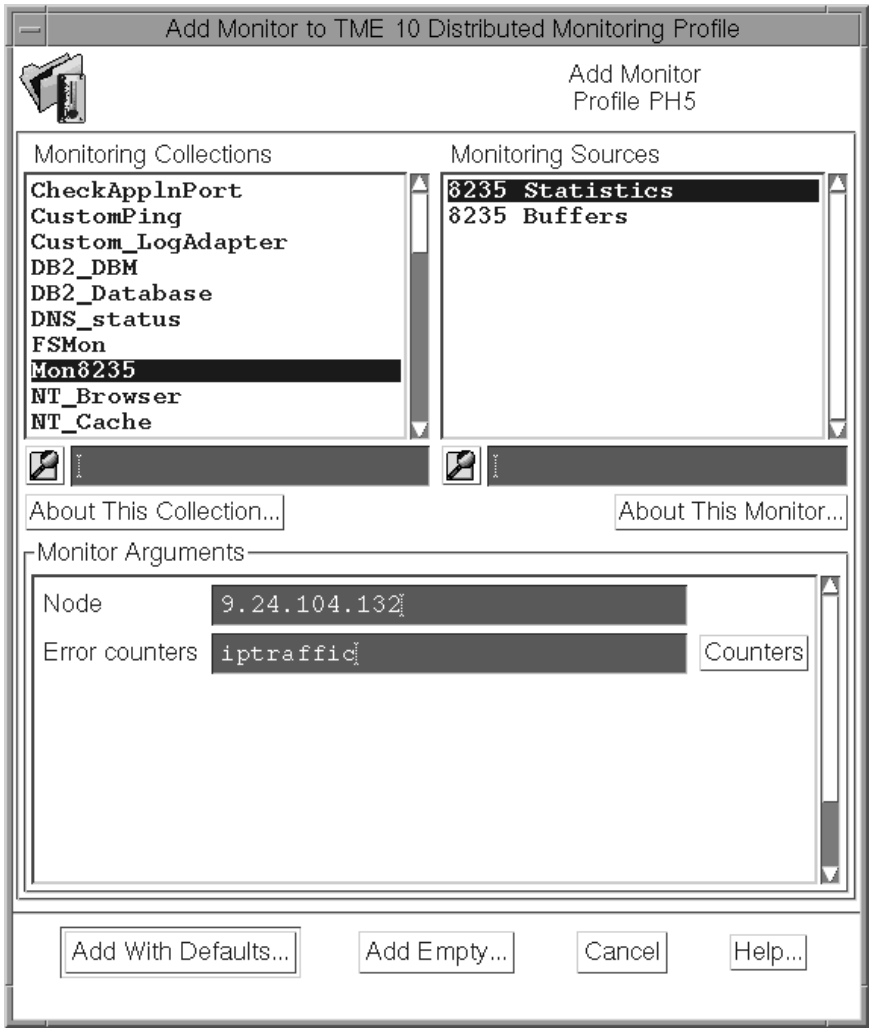


Figure 92. Mon8235 - Add Monitor

We start by adding a new profile PH5 to our profile manager, and we open PH5 and click on **Add Monitor...** The window in Figure 92 is displayed. We select the **Mon8235** monitoring collection and the **8235 Statistics** monitoring source. We know the IP address of the 8235 and fill in `9.24.104.132`. To display the possible choices for error counters we click on **Counters** and the window in Figure 93 on page 109 is displayed. We choose **IP traffic errors** and click on **Set & Close**.

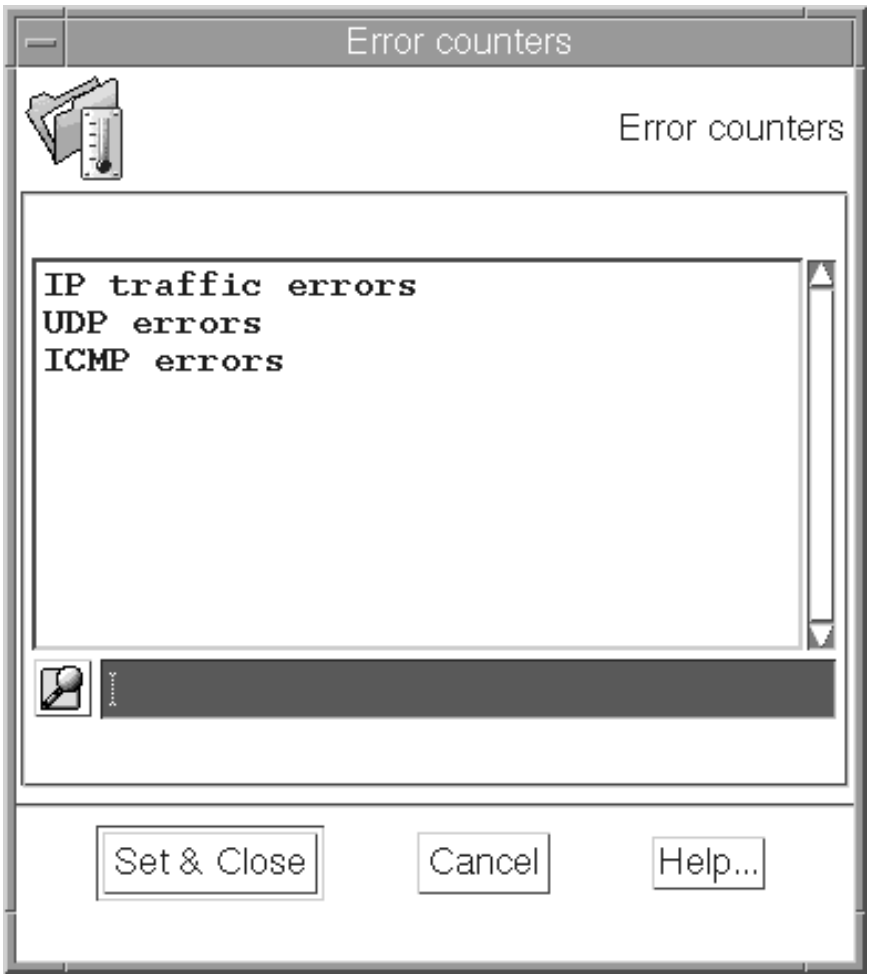


Figure 93. Mon8235 - Error Counters

The window in Figure 94 on page 110 shows up and we select Response level **always** and **Popup** to get an alert message every time the monitor fires. We set the monitoring schedule to every 2 minutes. These settings are only done for testing purposes; in a production environment you would of course only report on exceptions which would probably not be frequently.

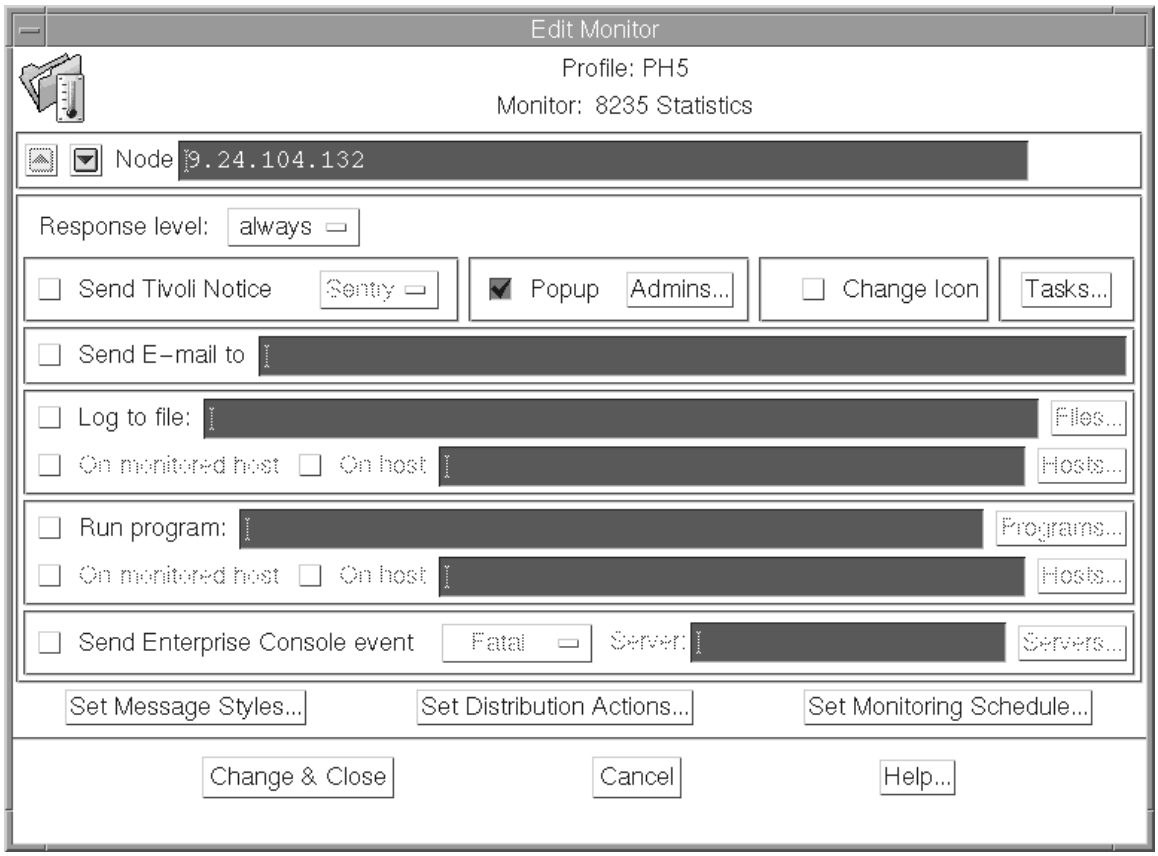


Figure 94. Mon8235 - Edit Monitor

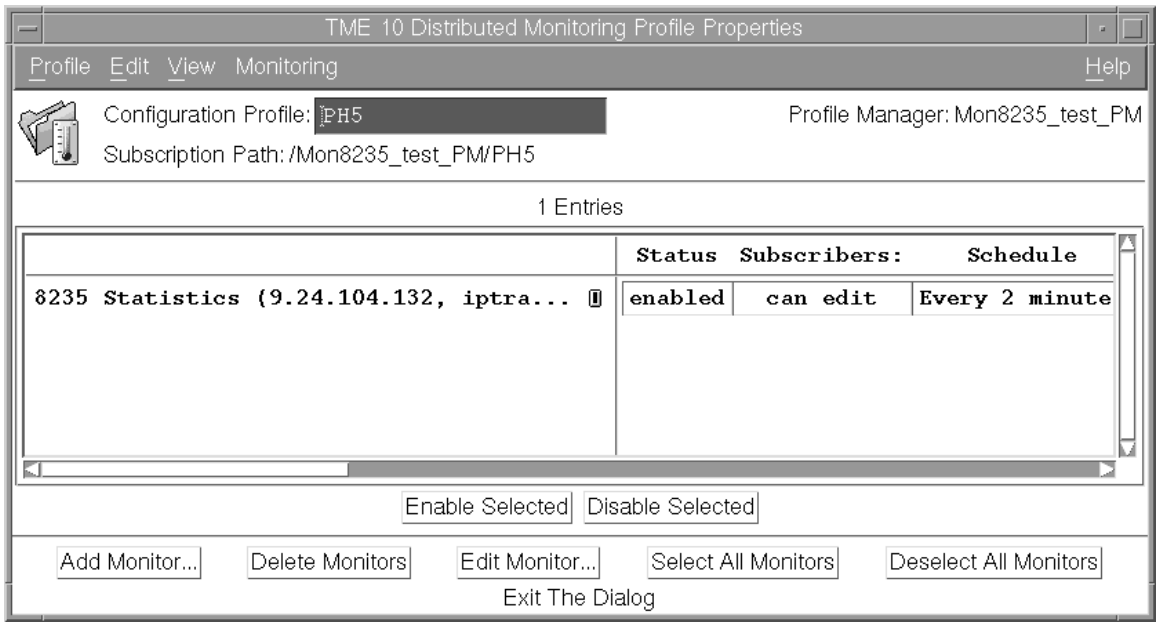


Figure 95. Mon8235 - Monitor Properties

In Figure 95 we select **Profile** from the menu bar and then **Save** from the pull-down menu. Then we select **Profile** from the menu bar again and then **Exit** from the pull-down menu.

Now we distribute the profile PH5 to the subscriber rs600019. We have already placed the Expect program and the mon8235 program in a directory that is in the $PATH.

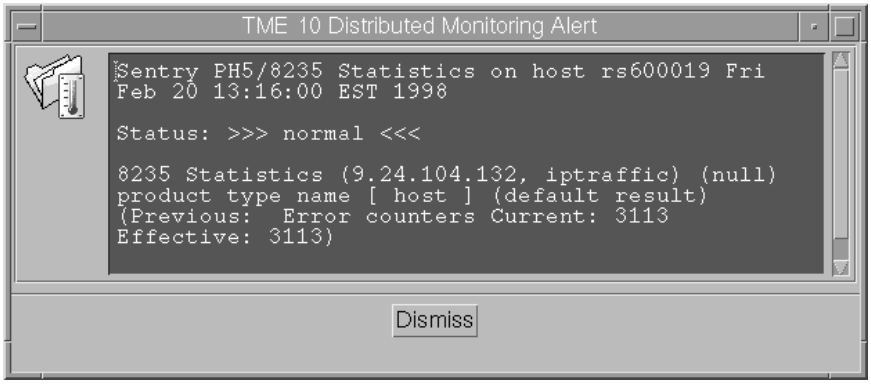


Figure 96. Mon8235 - Error Counters Pop-Up

In Figure 96 we see a pop-up window showing the IP traffic error counter being 3113. This is the field datagrams with header errors in Figure 86 on page 102.

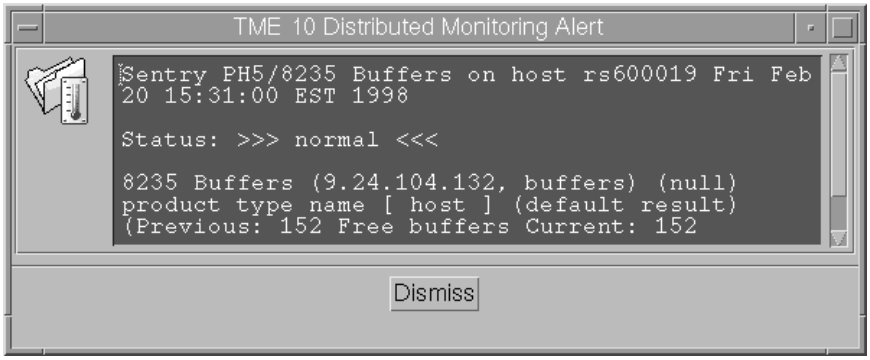


Figure 97. Mon8235 - Buffer Pop-Up

The 8235 Buffers monitor shows a pop-up window in Figure 97. The available number of buffers is 152. This value corresponds to the FREE field in Figure 85 on page 101.

3.10.3 Mon8235 - Summary

We have shown how to interact with a hardware box, by using a telnet session and interactively extract data from this session. There is much more data available on the IBM 8235 that can be monitored; you can even extract information down to modem level. Please refer to the appropriate IBM 8235 manuals for further information.

3.11 Monitoring Performance of a VM System

This monitoring collection will show that not only distributed systems and boxes can be monitored, but also mainframe systems.

To do this we decide to use the knowledge we acquired in monitoring the IBM 8235. Our target platform will be VM, but this technique could easily be extended to other platforms. VM is not a platform directly supported by Tivoli. Hence, in this

example we show that even these kind of platforms can be integrated into Tivoli management.

We decide to use the INDICATE command on VM which in the form we use it is a CP class G command, available to all users. Without operands it defaults to INDICATE LOAD and displays the following type of information:

- The percentage of usage for each processor in your system
- The usage of real storage
- Information concerning expanded storage
- Information concerning the minidisk cache
- The paging rate
- The number of users in the dispatch, eligible, and dormant lists
- The percentage of time that the system is executing instructions on the vector facility
- The number of recent vector facility users
- The number of real vector facilities that can be shared by users

A sample of the output from the INDICATE command is shown in Figure 98.

```
AVGPROC-017% 03 AVGVEC-000% 02
XSTORE-000013/SEC MIGRATE-0000/SEC
MDC READS-000172/SEC WRITES-000025/SEC HIT RATIO-090%
STORAGE-001% PAGING-0006/SEC STEAL-000%
Q0-00002 Q1-00003           Q2-00001 EXPAN-002 Q3-00002 EXPAN-002
```

Figure 98. Output from INDICATE Command on VM

An overview of our solution is shown in Figure 99 on page 113.

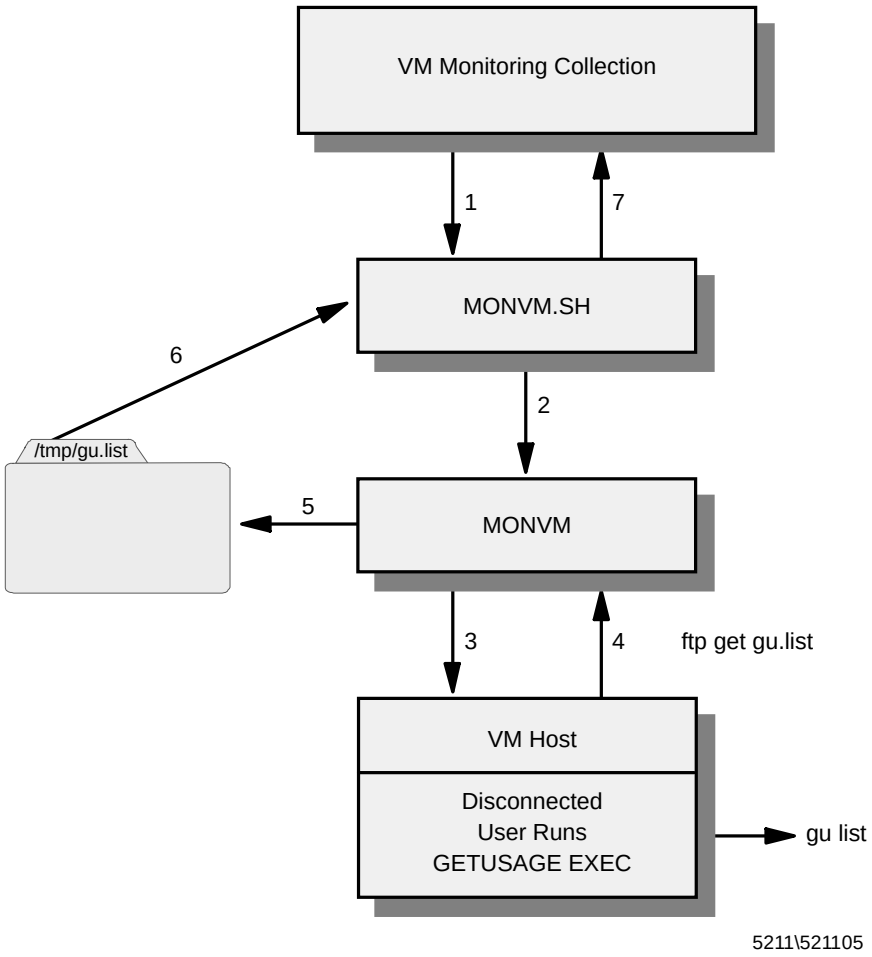


Figure 99. MonVM - Overview

The different steps are described below:

1. The monitor triggers the shell script monvm.sh.
2. The shell script monvm.sh calls monvm, written in Expect.
3. The monvm script finds the VM node user ID and password to use for ftp and then starts an ftp session with the VM node.
4. The file GU LIST, which is written every 60 seconds to the VM minidisk by a disconnected VM user ID, is transferred from VM to the Tivoli managed node. The file GU LIST is written by the GETUSAGE EXEC that is running on the VM system.

   VM allows a file transfer to a disconnected user, so the GETUSAGE EXEC can run simultaneously with the file transfer.
5. The monvm script writes the result to a file in /tmp and returns control to monvm.sh. The file is created as nodename.list, as we need a unique name for each instance of the monitor.
6. The monvm.sh script reads the nodename.list file and extracts performance data.
7. The data is returned to the monitor.

Our monitoring collection consists of the following elements:

- GETUSAGE EXEC
- monvm.users file in /etc/security
- monvm.msg
- monvm.csl
- monvm
- monvm.sh

3.11.1 MonVM Monitor Definitions

The first program we implement is the GETUSAGE EXEC on VM.

```
/* rexx */
Trace error2
milliseconds = 60000   /* 60 seconds */
Address CMS
Do forever
    'PIPE CP IND' '| > gu list a'
    call csl 'VMTHRDL rc re milliseconds'
End
exit
```

Figure 100. Listing of GETUSAGE EXEC for VM

To run this EXEC under a disconnected user you need to do the following:

1. Store the EXEC on the users mini-disk.
2. Type the following on the VM command line:

   ```
   set run on
   getusage
   #cp disc
   ```

If you want to halt execution at any time, just enter `HX`, which means halt execution in CMS terms.

The monvm.users file shown in Figure 101 consists of one row per target system with node, user ID, and password. The node name is the IP hostname of the VM system. The user IDs and passwords are stored in a file at a secure location in order to avoid having to specify them as arguments to the monitor where everybody could see them.

```
wtscpok.itso.ibm.com pholm xxxxxxx
stovm1.sto.se.ibm.com pholm xxxxxxx
xxxxxx.xxx.xx.ibm.com usera passa
```

Figure 101. Listing of monvm.users File

Note

In the above example file, the actual passwords have been replaced with 'x' characters.

The next thing we need is the message file.

```
$ key=monvm_Descr
1 VM Performance
$ key=monvm_Help
2 This Monitor will extract information from a VM system
using the indicate command. This is done by ftp to the VM system and
getting a previously stored file. This file is stored by
a REXX exec that runs under a disconnected user.
$ key=monvm_val_Descr
3 Performance
$ key=monvm_node
4 Node
$ key=monvm_command
5 Performance info
$ key=monvm_avgproc
6 Average Processor Usage
$ key=monvm_xstore
7 Expanded Storage Usage
$ key=monvm_reads
8 Reads/SEC
$ key=monvm_writes
9 Writes/SEC
$ key=monvm_storage
10 Storage used
$ key=monvm_paging
11 Paging/SEC
$ key=monvm_steal
12 Steal
$ key=monvm_ratio
13 Hit Ratio
$ key=monvm_q0
14 Q0
$ key=monvm_q1
15 Q1
$ key=monvm_q2
16 Q2
$ key=monvm_q3
17 Q3
$ key=ButtonLabelChoice
18 Type of information
$ key=generic_help
19 The monitor will extract information from a VM system.
For more information please use the Redbook 24-5211,
Creating Custom Monitoring Collections.
```

Figure 102. Message Catalog for VM Monitor (monvm.msg)

The message file shown in Figure 102 defines a button for different performance selections. The button with its label `Type of Information` is defined in item 18. The different selections are defined in items 6 through 17.

The next figure shows the MCSL definitions for our VM monitor.

```
#include "Sentry2_0.dsl"

#include "monvm.dsl"

Collection "MonVM" {
    CodeID = "$Id:monvm.csl,v 1.0$";
    Version = "1.0";
    Require = ">2.0.2";
    HelpMessage = (monvm_generic_help);
    EventBaseClass = "Sample_Sentry_Monitors";
    NoticeGroup = "Sentry";

#include "operators.csl"
#include "formats.csl"
#include "choicelists.csl"

    ChoiceList Info {
      ButtonLabel = (monvm_ButtonLabelChoice);
      {
         { (monvm_monvm_avgproc) "avgproc" }
         { (monvm_monvm_xstore) "xstore" }
         { (monvm_monvm_reads) "reads" }
         { (monvm_monvm_writes) "writes" }
         { (monvm_monvm_storage) "storage" }
         { (monvm_monvm_paging) "paging" }
         { (monvm_monvm_ratio) "ratio" }
         { (monvm_monvm_steal) "steal" }
         { (monvm_monvm_q0) "q0" }
         { (monvm_monvm_q1) "q1" }
         { (monvm_monvm_q2) "q2" }
         { (monvm_monvm_q3) "q3" }
       };
    };

    Monitor Performance Numeric Group numeric{
      Description = (monvm_monvm_Descr);
      ValueDescription = (monvm_monvm_val_Descr);
      HelpMessage = (monvm_monvm_Help);
      Argument (monvm_monvm_node)
        DefaultValue "";
      Argument (monvm_monvm_command)
        RestrictedChoice "Info"
        DefaultValue "avgproc";
      Implementation (aix4-r1)
      Shell("/bin/ksh", "-c", Command, "MONVM")
      Import "monvm.sh";
   };
}
```

Figure 103. Listing of monvm.csl

The csl file listed in Figure 103 defines one monitor called Performance with two arguments, the node and the command option.

The monvm script, written in Expect, is the main driver of our monitor.

```
#!/usr/local/bin/expect --

if {[llength $argv] < 1} {
      puts "Usage is monvm address "
      exit 1
     }
set node [lindex $argv 0]
set cmd  [lindex $argv 1]
set env(TERM) vt100
set user ""
set file "/etc/security/monvm.users"

# Disable stdout output
log_user 0

# Find out userid and password for the vm system
set fp [open $file r]

while {[gets $fp line] != -1} {
        scan $line "%s %s %s" wnode wuser wpass
        if {![string compare $wnode $node]} {
          set user $wuser
          set pass $wpass
        }
      }

close $fp

if {$user == ""} {
      puts "No matching nodeid found in monvm.users file"
      exit 1
     }
# Start ftp session
spawn ftp $node

# Uncomment line below to get debug information to stderr
#exp_internal 1

# Login to ftp
expect ": "
send "$user\r"
expect "Password: "
send "$pass\r"

expect "ftp> "

send "get gu.list /tmp/$node.list\r"
expect "ftp> "
# Logout from ftp
send "quit\r"

# Enable stdout output
log_user 1

exit 0
```

Figure 104. Listing of monvm Script

The monvm script first reads the user ID and password to be used from the monvm.users file and then performs an automatic file transfer to the VM system. Once the ftp session is established, the file gu.list is downloaded from the VM system.

```
#!/bin/ksh
monvm $1
  case $2 in
    "avgproc")  vmval=$(grep "AVGPROC" /tmp/$1.list | cut -d'-' -f2 | cut -c1-3);;
    "xstore")  vmval=$(grep "XSTORE" /tmp/$1.list | cut -d'-' -f2 | cut -c1-6);;
    "reads")  vmval=$(grep "READS" /tmp/$1.list | cut -d'-' -f2 | cut -c1-6);;
    "writes")  vmval=$(grep "WRITES" /tmp/$1.list | cut -d'-' -f3 | cut -c1-6);;
    "ratio")  vmval=$(grep "RATIO" /tmp/$1.list | cut -d'-' -f4 | cut -c1-3);;
    "storage")  vmval=$(grep "STORAGE" /tmp/$1.list | cut -d'-' -f2 | cut -c1-3);;
    "paging")  vmval=$(grep "PAGING" /tmp/$1.list | cut -d'-' -f3 | cut -c1-4);;
    "steal")  vmval=$(grep "STEAL" /tmp/$1.list | cut -d'-' -f4 | cut -c 1-3);;
    "q0")  vmval=$(grep "Q0" /tmp/$1.list | cut -d'-' -f2 | cut -c1-5);;
    "q1")  vmval=$(grep "Q1" /tmp/$1.list | cut -d'-' -f3 | cut -c1-5);;
    "q2")  vmval=$(grep "Q2" /tmp/$1.list | cut -d'-' -f4 | cut -c1-5);;
    "q3")  vmval=$(grep "Q3" /tmp/$1.list | cut -d'-' -f6 | cut -c1-5);;
  esac
echo $vmval
exit
```

Figure 105. Listing of monvm.sh Script

The monvm.sh script starts monvm, passing on the first parameter that has been passed to itself by the monitor. Then, depending on the second parameter that has been passed from the monitor, the appropriate performance parameter is parsed from the file that is downloaded by the monvm script.

3.11.2 Running the MonVM Monitor Collection

Before starting with this, we need to have done the following:

- Installed and started the GETUSAGE EXEC on a VM user ID.
- Distributed the Expect program and monvm program to the node where we will run the Tivoli Distributed Monitoring monitor.
- Edited a monvm.users file and placed it in /etc/security.

We create a new profile PH7 and open it to add a new monitor. The window in Figure 106 on page 119 is displayed.

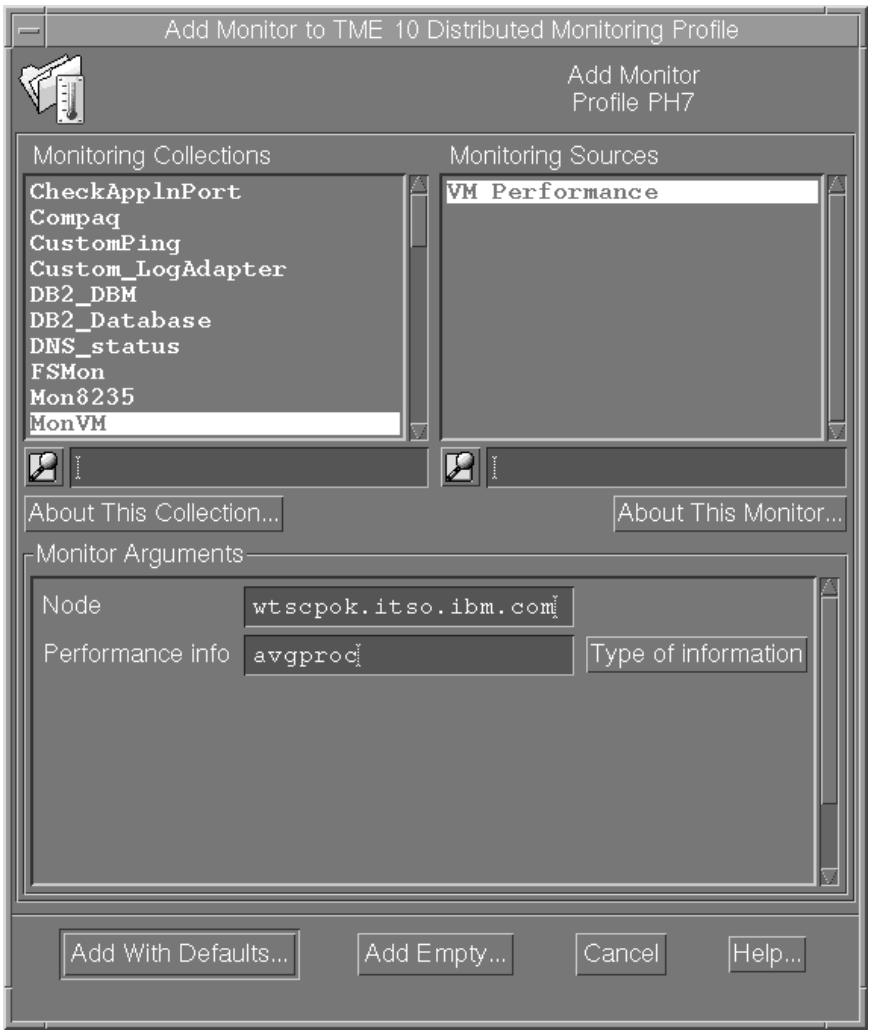


Figure 106. MonVM - Add Monitor

We select **MonVM** from the Monitoring Collections box and then **VM Performance** in the Monitoring Sources section. We fill in the Node argument as `wtscpok.itso.ibm.com` and click on **Type of Information** to select the second parameter. The window in Figure 107 on page 120 is displayed. We select **Reads/SEC** and click on **Set & Close**.

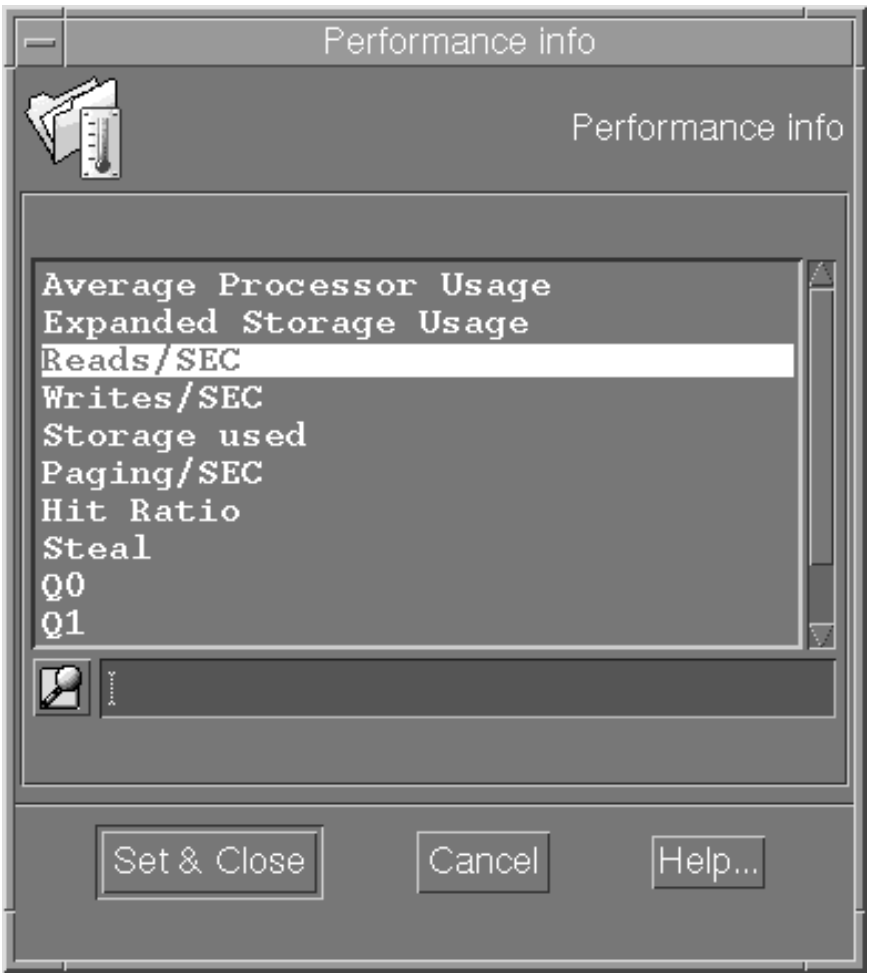


Figure 107. MonVM - Performance Info

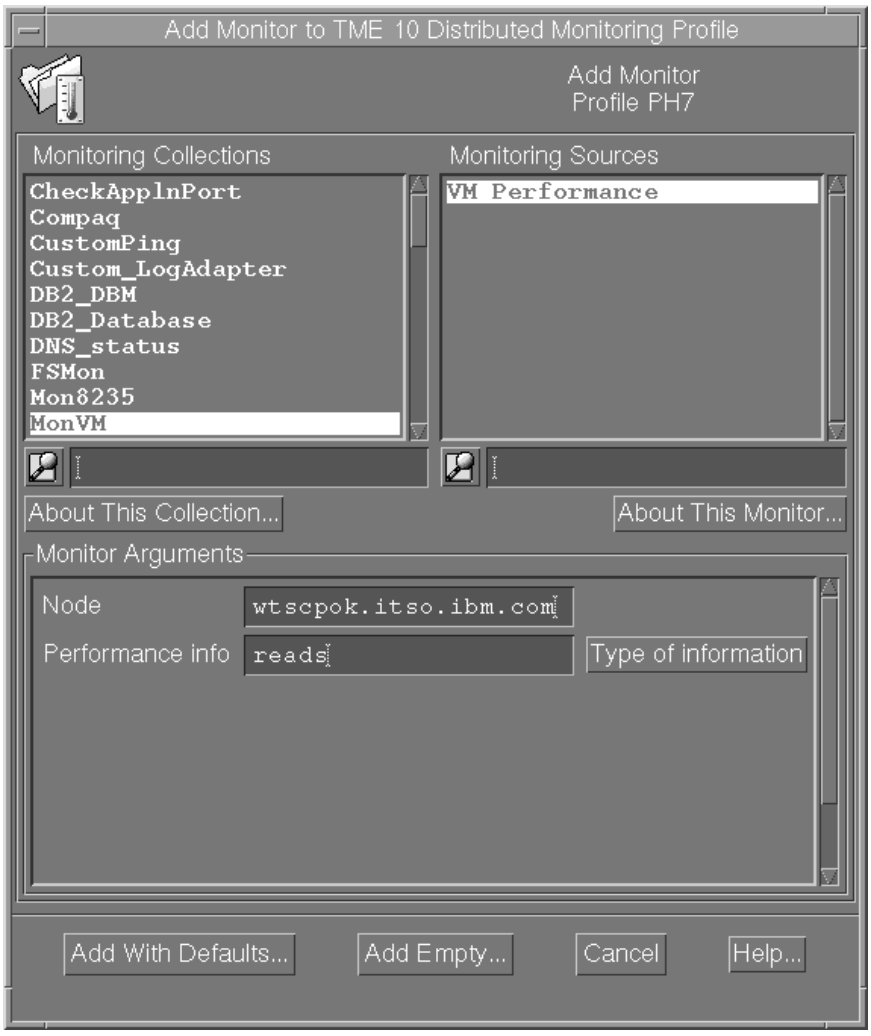


Figure 108. MonVM - Add Monitor with Changed Argument

In Figure 108 we now have the selected `reads` value filled in as Performance info and we click on **Add Empty...**.

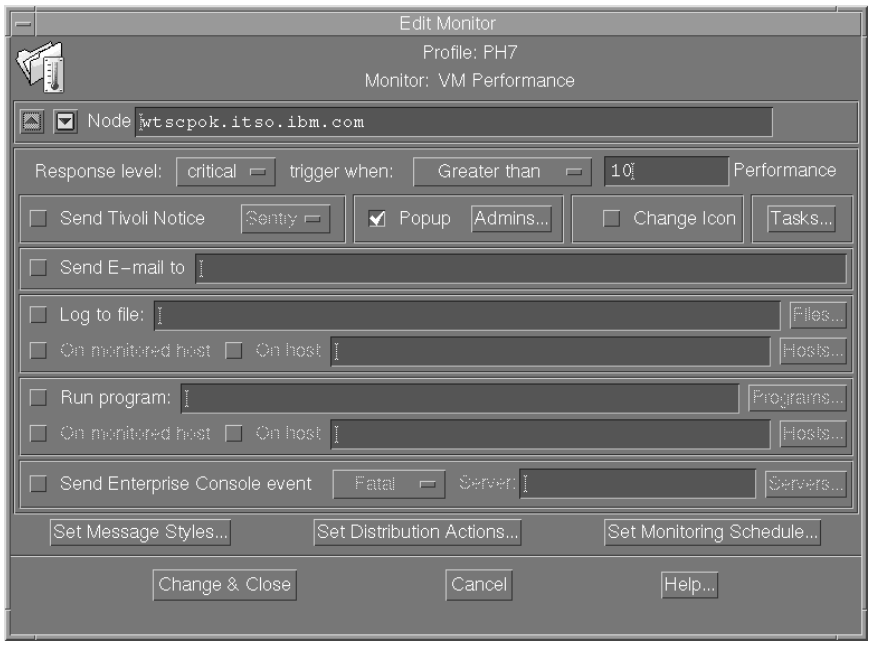


Figure 109. MonVM - Edit Monitor

In Figure 109 we select Response level **Critical** and **Greater than** and fill in `10`. We click **Popup...** and select the Root administrator and then click on **Change & Close**.

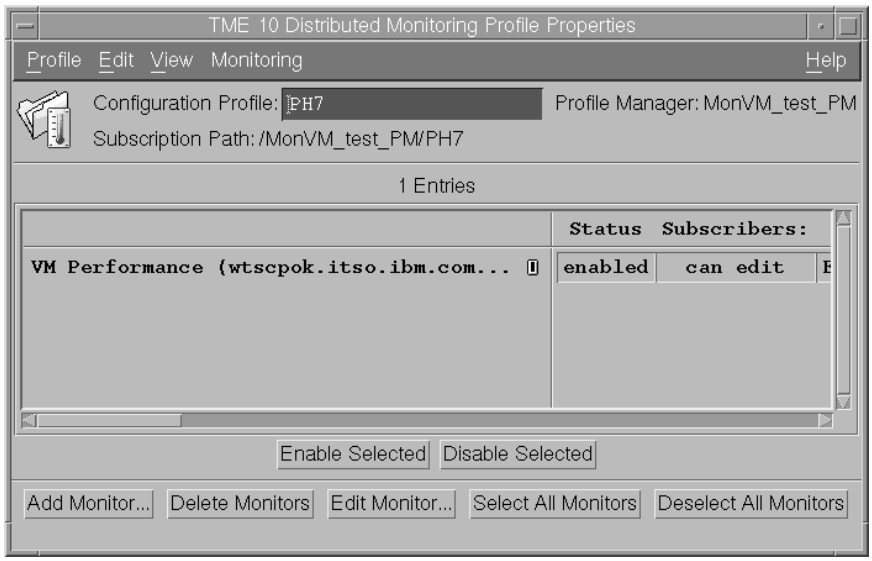


Figure 110. MonVM - Profile Properties

We select **Profile** from the menu bar and then select **Save** from the pull-down menu. Then we select **Profile** from the menu bar again and then **Close** from the pull-down menu. We distribute the profile to rs600019, where we have already placed Expect and the monvm program in the path.

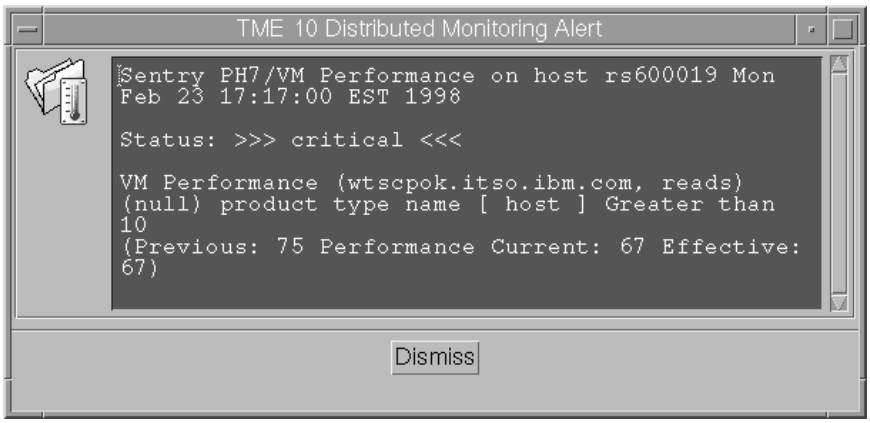


Figure 111. MonVM - Critical Alert

The pop-up window in Figure 111 shows a critical alert, where the threshold for reads was set to 10, and the current value is 67. You also see the hostname, which is wtscpok.itso.ibm.com, and that the previous value was 75.

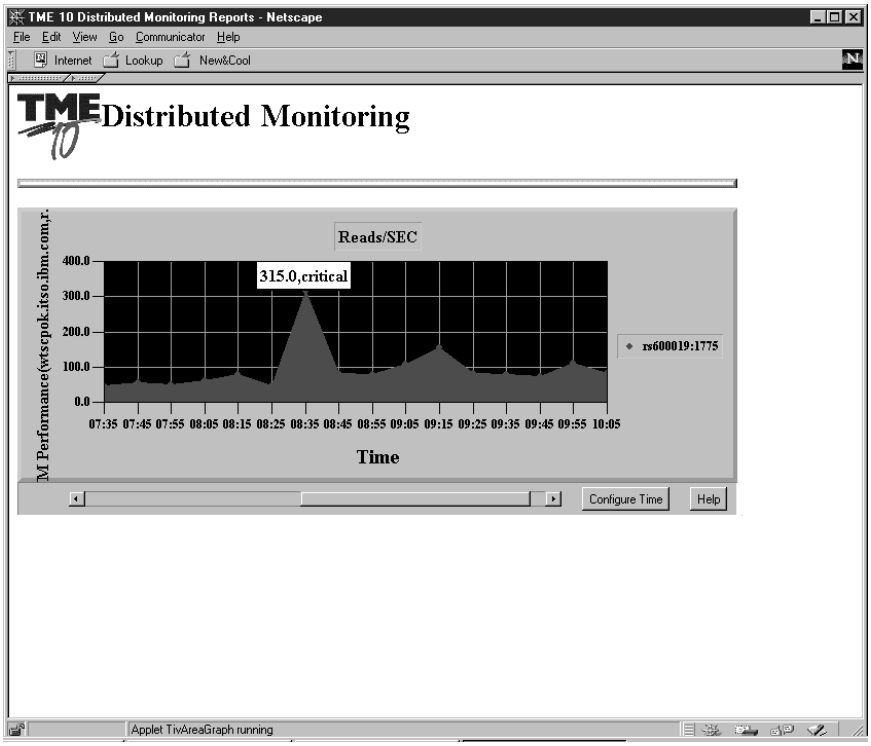


Figure 112. MonVM - WEB Interface

In Figure 112 we show the Web interface for the VM Performance Monitor. In order to create this Web output you need to set the response level to always and select the Create graphable log task from the Sentry Graphable Logs collection to run every time the monitor begins. This will log the output to a file that is used to create the Web graph.

For more information on this topic, refer to the redbook *A First Look at TME 10 Distributed Monitoring 3.5*, SG24-2112.

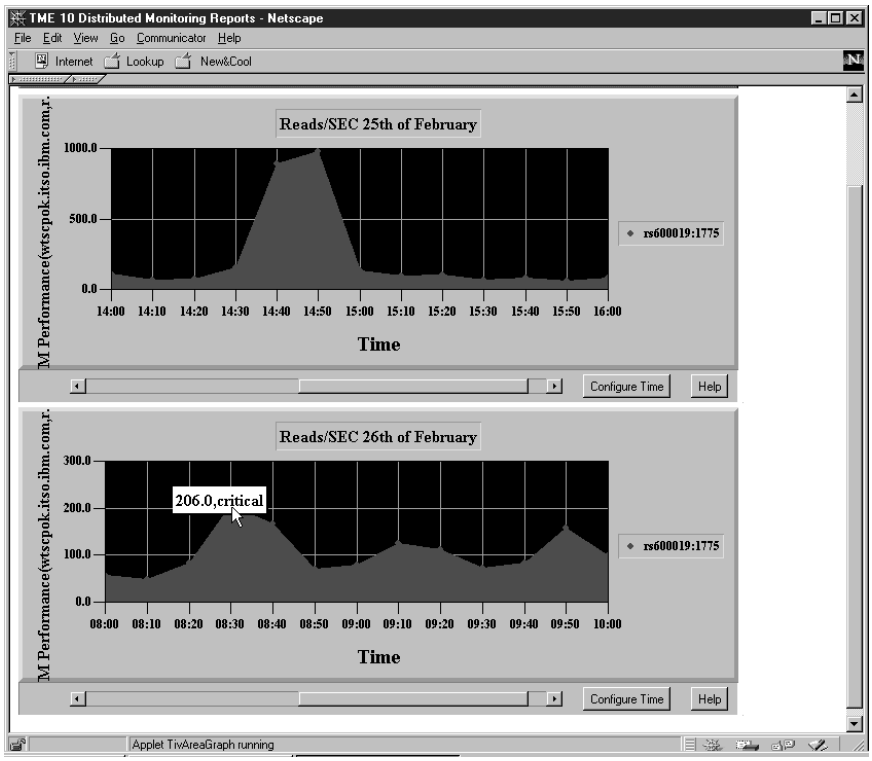


Figure 113. MonVM - WEB Interface Double Charts

In Figure 113 two different days are displayed. Please note that the actual value on the chart will be displayed when you place the cursor over the chart, in our case `206.0, critical`.

3.11.3 MonVM - Summary

We have shown that by using a VM platform-specific monitor, we can intercept that data and use it under Tivoli Distributed Monitoring. This technique can be applied to many other platforms, where there today is no support for Tivoli yet.

3.12 Monitoring DB2

We want to monitor some data for DB2. Although there are Tivoli modules available to specifically manage RDBMSs, we want to show a simple example of how you can get started writing your own RDBMS monitors.

The monitor we create works similar to the Oracle and Sybase monitoring collections that are provided with Tivoli Distributed Monitoring.

The tool we decide to use to gather DB2 information is the DB2 Snapshot Monitor, a standard component of DB2. The Snapshot Monitor is described in detail in *DB2 Database System Monitor Guide and Reference*, S20H-4871. DB2 is installed on our TMR server rs600019 for use with Tivoli Inventory. Tivoli Inventory is used to scan hardware data on a number of UNIX boxes.

We first have a look at the DB2 Snapshot Monitor and examine its output and then write a custom monitor to process this output and include it in Tivoli Distributed Monitoring.

3.12.1 DB2 Setup

We use DB2 Version 4.1, and have the DB2 administrator ID set to dbmsadm. The DB2 administrator is the instance owner and has the highest database priority. TME 10 Inventory 3.2 is installed and the database name is set to tiv_inv. We just use TME 10 Inventory 3.2 as a sample RDBMS application that creates some load on the RDBMS.

To have the DB2 environment activated every time you log in as dbmsadm, add the following to the .profile file for the dbmsadm user:

```
. ./sqllib/db2profile
```

A list of installed DB2 components is shown in Figure 114.

```
#lslpp -l | grep db2

db2_02_01.client           4.1.1.2    C    DB2 Client Application Enabler
db2_02_01.clp              4.1.1.2    C    DB2 Command Line Processor
db2_02_01.conv             4.1.1.2    C    DB2 Code Page Conversions
db2_02_01.cs.rte           4.1.1.2    C    DB2 Communications Support -
                                           Base with TCPIP
db2_02_01.db2.misc         4.1.1.2    C    DB2 Utilities and Samples
db2_02_01.db2.rte          4.1.1.2    C    DB2 Executables
db2_02_01.dd               4.1.1.2    C    DB2 Database Director
db2_02_01.doc.En_US.ipfx   4.1.1.2    C    DB2 Product Library - INF -
                                           En_US
db2_02_01.msg.en_US        4.1.1.2    C    DB2 Product Messages - en_US
db2_02_01.pm               4.1.1.2    C    DB2 Performance Monitor
db2_02_01.sdk.cli          4.1.1.2    C    DB2 Call Level Interface Sample
db2_02_01.sdk.misc         4.1.1.2    C    DB2 SDK Utilities and Samples
db2_02_01.ve               4.1.1.2    C    DB2 Visual Explain
```

Figure 114. Installed DB2 Components

3.12.2 Snapshot Monitor Setup

The Snapshot Monitor will give you information on different levels:

- Database manager
- Database
- Application
- Table
- Locks
- Table spaces

All levels with the exception of the database manager require that you set the monitoring switches with the Update monitoring switches command to get information. By default the switches are turned off. To see what settings you currently have, issue the following command as the dbmsadm user:

```
db2 get monitor switches
```

The output from the command is shown in Figure 115 on page 126.

```
$ db2 get monitor switches

            Monitor Recording Switches

Buffer Pool Activity Information (BUFFERPOOL) = OFF
Lock Information                        (LOCK) = OFF
Sorting Information                     (SORT) = OFF
SQL Statement Information           (STATEMENT) = OFF
Table Activity Information              (TABLE) = OFF
Unit of Work Information                  (UOW) = OFF

$
```

Figure 115. Output from db2 get monitor switches Command

To make the settings permanent we use the following DB2 commands:

```
db2 update dbm cfg using DFT_MON_LOCK ON
db2 update dbm cfg using DFT_MON_BUFPOOL ON
```

A restart of DB2 is necessary to pick up the new settings. This is done with `db2stop` and `db2start`. The output from the get monitor switches command after the update is shown in Figure 116.

```
$ db2 get monitor switches

            Monitor Recording Switches

Buffer Pool Activity Information (BUFFERPOOL) = ON  02-09-1998 13:56:00.000109
Lock Information                        (LOCK) = ON  02-09-1998 13:56:00.000109
Sorting Information                     (SORT) = OFF
SQL Statement Information           (STATEMENT) = OFF
Table Activity Information              (TABLE) = OFF
Unit of Work Information                  (UOW) = OFF

$
```

Figure 116. Output from db2 get monitor switches Command After Update

Note

We found out that if we use the database manager level we always get data back, but if we use another level data was not always available. In our case this is probably due to very little system activity.

3.12.3 Sample Output from Snapshot Monitor

Some output from Snapshot Monitor is shown in Figure 117 on page 127.

```
            Database Manager Snapshot

Node type                                  = Database Server with local
                                             and remote clients
Instance name                              = dbmsadm
Database manager status                    = Active

Sort heap allocated                        = 0
Post threshold sorts                       = Not Collected
Piped sorts requested                      = 55
Piped sorts accepted                       = 55

Start Database Manager timestamp           = 02-05-1998 17:47:00.000127
Last reset timestamp                          =
Snapshot timestamp                         = 02-03-2008 17:46:59.999999

Remote connections to db manager           = 0
Remote connections executing in db manager = 0
Local connections                          = 0
Local connections executing in db manager  = 0
Active local databases                     = 0

High water mark for agents registered      = 4
High water mark for agents waiting for token= 0
Agents registered                          = 4
Agents waiting for a token                 = 0
Idle agents                                = 2

Committed private Memory (Bytes)           = 2195456
```

Figure 117. Output from db2 get snapshot from dbm Command

3.12.4 Database Manager Monitor

Now that we know the input data, we can implement our monitor. The monitor itself is implemented as a shell script. We use MCSL to create a monitoring collection. We start with the MCSL files and then show the actual shell script implementation of the monitor.

This monitoring collection consists of two monitors, one monitoring counters and the other one monitoring memory. The following files are the elements of the monitoring collection, DB2_DBM:

- snapm.msg

  This is the message file for the monitoring collection.

```
$ key=snap1_Descr
1 Count
$ key=snap1_Help
2 This Monitor returns the current value on a selected counter from the
output of the DB2 Snapshot Monitor.
$ key=snap1_val_Descr
3 Count
$ key=snap1_list
4 Options
$ key=Remote_conn
5 Remote connections to db manager
$ key=Remote_conn_exec
6 Remote connections executing in db manager
$ key=Local_conn
7 Local connections
$ key=Local_conn_exec
8 Local connections executing in db manager
$ key=Active_local
9 Active local databases
$ key=High_wm_reg
10 High water mark for agents registered
$ key=High_wm_wait
11 High water mark for agents waiting for a token
$ key=Agents_reg
12 Agents registered
$ key=Agents_wait
13 Agents waiting for a token
$ key=Agents_idle
14 Idle agents
$ key=generic_help
15 The monitor extracts counters from the snapshot output. For more
information please use the DB2 Database System Monitor Guide
and Reference.
$ key=ButtonLabelChoice
16 Choices
$ key=snap2_val_Descr
17 Bytes
$ key=snap2_Descr
18 Bytes
$ key=snap2_Help
19 This Monitor returns the number of bytes Committed for Private Memory
output of the DB2 Snapshot Monitor.
$ key=DBM_id
20 DBM admin id
$ key=Piped_sort
21 Difference piped sort
```

Figure 118. snapm.msg Listing

- snapm.csl

  This is the csl file and it defines two monitors, Counters and Memory. The first argument for both monitors is the DBM admin ID. For the Counters monitor there is a ButtonLabelChoice function, which will show a number of possible selections for the second argument.

```
#include "Sentry2_0.dsl"
#include "snapm.dsl"

Collection "DB2_DBM" {
    CodeID = "$Id:snapm.csl,v 1.0$";
    Version = "1.0";
    Require = ">2.0.2";
    HelpMessage = (snapm_generic_help);
    EventBaseClass = "Sample_Sentry_Monitors";
    NoticeGroup = "Sentry";
#include "operators.csl"
#include "choicelists.csl"
#include "formats.csl"

    ChoiceList DiffOptions {
      ButtonLabel = (snapm_ButtonLabelChoice);
      {
         { (snapm_Remote_conn) "Remote_conn" }
         { (snapm_Remote_conn_exec) "Remote_conn_exec" }
         { (snapm_Local_conn) "Local_conn" }
         { (snapm_Local_conn_exec) "Local_conn_exec" }
         { (snapm_Active_local) "Active_local" }
         { (snapm_High_wm_reg) "High_wm_reg" }
         { (snapm_High_wm_wait) "High_wm_wait" }
         { (snapm_Agents_reg) "Agents_reg" }
         { (snapm_Agents_wait) "Agents_wait" }
         { (snapm_Agents_idle) "Agents_idle" }
         { (snapm_Piped_sort) "Piped_sort" }
      };
    };

    Monitor Counters Numeric Group numeric {
      Description = (snapm_snap1_Descr);
      ValueDescription = (snapm_snap1_val_Descr);
      HelpMessage = (snapm_snap1_Help);
      Argument (snapm_DBM_id)
        DefaultValue "dbmsadm";
      Argument (snapm_snap1_list)
        RestrictedChoice "DiffOptions"
        DefaultValue "Remote_conn";
      Implementation (aix3-r2, aix4-r1)
      Shell("/bin/ksh", "-c", Command, "SNAPM1")
      Import "snapm1.sh"
     ;
    };
```

Figure 119 (Part 1 of 2). smapm.csl Listing

```
    Monitor Memory Numeric Group numeric {
      Description = (snapm_snap2_Descr);
      ValueDescription = (snapm_snap2_val_Descr);
      HelpMessage = (snapm_snap2_Help);
      Argument (snapm_DBM_id)
        DefaultValue "dbmsadm";
      Argument (snapm_committed)
        Restriction "Committed_memory"
        DefaultValue "Committed_memory";
      Implementation (aix3-r2, aix4-r1)
      Shell("/bin/ksh", "-c", Command, "SNAPM2")
      Import "snapm2.sh"
     ;
    };
}
```

Figure 119 (Part 2 of 2). smapm.csl Listing

Note

The first monitor definition in Figure 119 on page 129 uses RestrictedChoice to force values from a list, in this case DiffOptions. The second monitor definition in Figure 119 on page 129 uses Restriction to force a specific value.

- snapm1.sh

  This is the shell script being called by the Counters monitor. It will look for specific strings in the output from the Snapshot Monitor. The only calculation being done is the difference between Piped sort requested and Piped sort accepted. The difference should normally be 0; if not tuning might be needed. Please refer to *DB2 Database System Monitor Guide and Reference*, S20H-4871 for more information.

  Notice that we run the db2 command in the script as the DBM admin, which is specified as the first parameter to the script.

```
#!/bin/ksh
#
# snapm1.sh
# sample monitor for DB2 for AIX
# that will analyze the output from snapshot,
# regarding different counters for DBM.
#
# Peter Holm, PHOLM@SE.IBM.COM
# ITSO Raleigh 1998
#
#

if [ "$1" = "" ]
then
  print "Specify DB admin's ID as first parameter."
  exit 1
fi
#
# get snapshot
#
su - $1 -c "db2 get snapshot for dbm" >/tmp/snapm

case $2 in
 "Remote_conn" )
       m=$(grep "Remote connections to db" /tmp/snapm | cut -d'=' -f2);;
 "Remote_conn_exec" )
       m=$(grep "Remote connections execut" /tmp/snapm | cut -d'=' -f2);;
 "Local_conn" )
       m=$(grep "Local connections" /tmp/snapm | cut -d'=' -f2);;
 "Local_conn_exec" )
       m=$(grep "Local connections execut" /tmp/snapm | cut -d'=' -f2);;
 "Active_local" )
       m=$(grep "Active local databases" /tmp/snapm | cut -d'=' -f2);;
 "High_wm_reg" )
  m=$(grep "High water mark for agents reg" /tmp/snapm | cut -d'=' -f2);;
 "High_wm_wait" )
 m=$(grep "High water mark for agents wait" /tmp/snapm | cut -d'=' -f2);;
 "Agents_reg" )
       m=$(grep "Agents registered" /tmp/snapm | cut -d'=' -f2);;
 "Agents_wait" )
 token" /tmp/snapm | cut -d'=' -f2);;
 "Agents_idle" )
       m=$(grep "Idle agents" /tmp/snapm | cut -d'=' -f2);;
 "Piped_sort" )
       m1=$(grep "Piped sorts requested" /tmp/snapm | cut -d'=' -f2);
       m2=$(grep "Piped sorts accepted" /tmp/snapm | cut -d'=' -f2);
       m=$(($m1-$m2));;
esac

echo $m
exit 0
```

Figure 120. snapm1.sh Listing

- snapm2.sh

This is the shell script being called by the Memory monitor. It will look for a specific string in the output from the Snapshot Monitor.

```
#!/bin/ksh
# snap2.sh
# sample monitor for DB2 for AIX
# that will analyze the output from snapshot,
# regarding Committed Private Memory.
#
# Peter Holm
# ITSO Raleigh 1998
#
#

if [ "$1" = "" ]
then
  print "Specify DB admin's ID as first parameter."
  exit 1
fi
#
# get snapshot
#
su - $1 -c "db2 get snapshot for dbm" >/tmp/snap2

m=$(grep "Committed private" /tmp/snap2 | cut -d'=' -f2)
echo $m

exit 0
```

Figure 121. Listing of snapm2.sh

To create the monitoring collection we issue the following commands:

1. `gencmsg snapm.msg`

   to compile the message file for snapm.

2. `mcsl -P ./cpp -x ./snapm.col snapm.csl -lang-c++ nostdinc undef`

   to compile the snapm monitoring collection.

3. `mcsl -iR snapm.col`

   to install the snapm monitoring collection.

To make your changes effective you need to recycle oserv. This is done with:

```
odadmin reexec all
```

3.12.5 Running the Database Manager Monitor

After doing the necessary compilations, we now have a ready monitoring collection. Let's see what it looks like.

We start with defining a profile manager and a Tivoli Distributed Monitoring profile. We open the profile and click on **Add Monitor...** and the following window is displayed:

Figure 122. snapm1 - Add Monitor

We first select the **DB2_DBM** monitoring collection and then we select the **Count** monitoring source and the Monitoring Arguments section shows up as seen in Figure 123 on page 134.

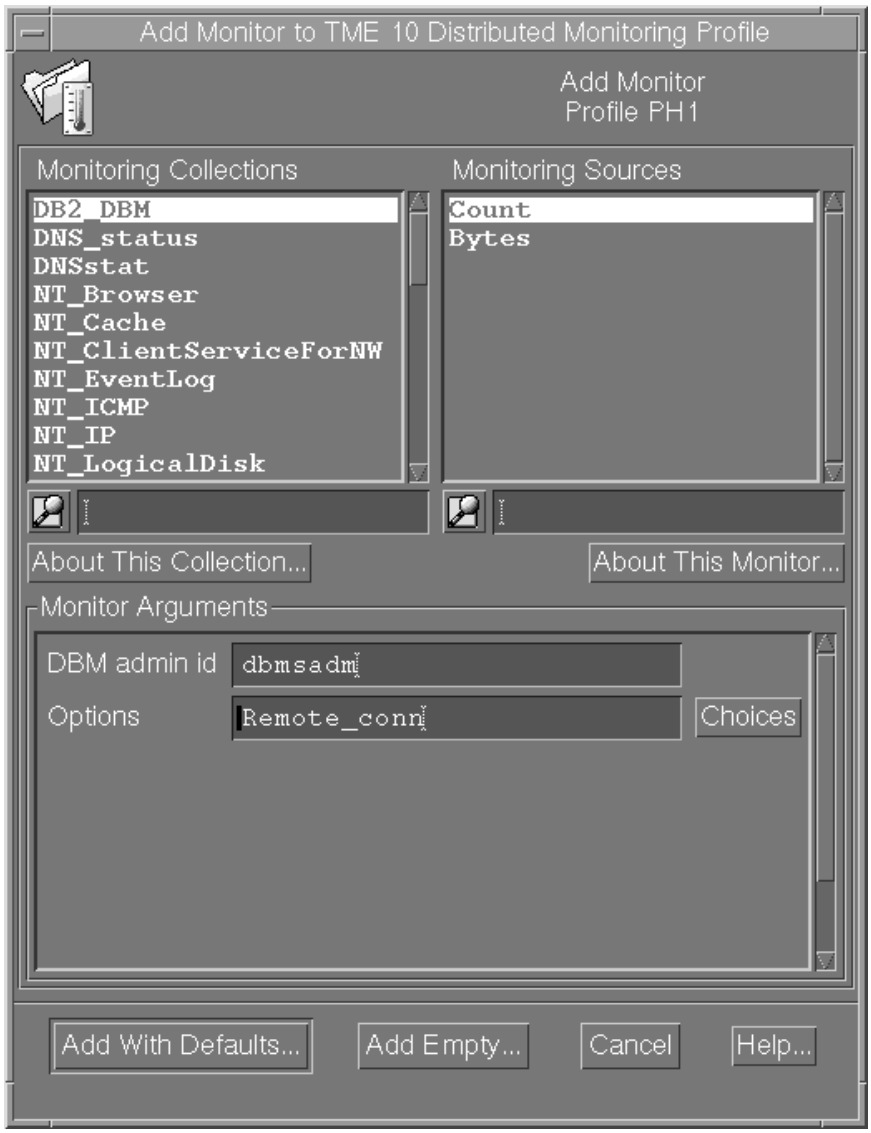


Figure 123. snapm1 - Arguments

We want to monitor the High water mark for agents registered, and click on **Choices** and the window in Figure 124 on page 135 is displayed.

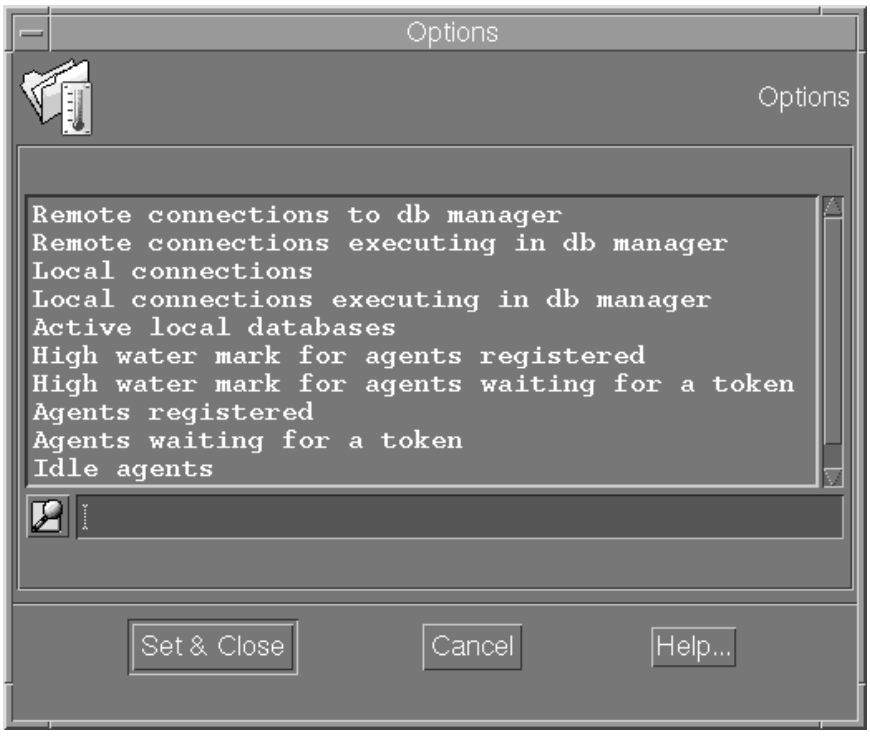


Figure 124. snapm1 - Options

This is the list of options we created by using the ChoiceList function in the snapm.csl, which is shown in Figure 119 on page 129. From the Options list we select the desired value, in our example **High water mark for agents registered**. We then select **Set & Close**.

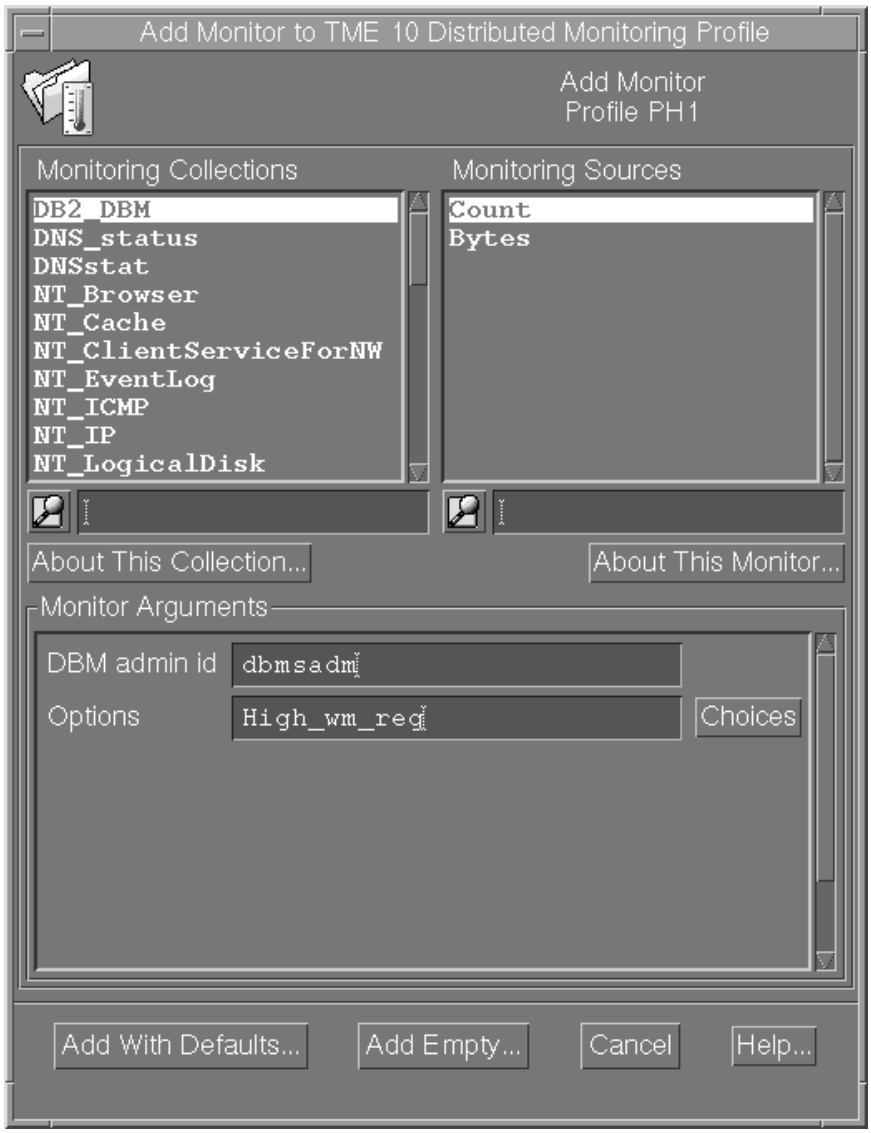


Figure 125. snapm1 - Add Empty

Verify that the monitor arguments are correctly specified. We select **Add Empty...** and the window in Figure 126 on page 137 is displayed.

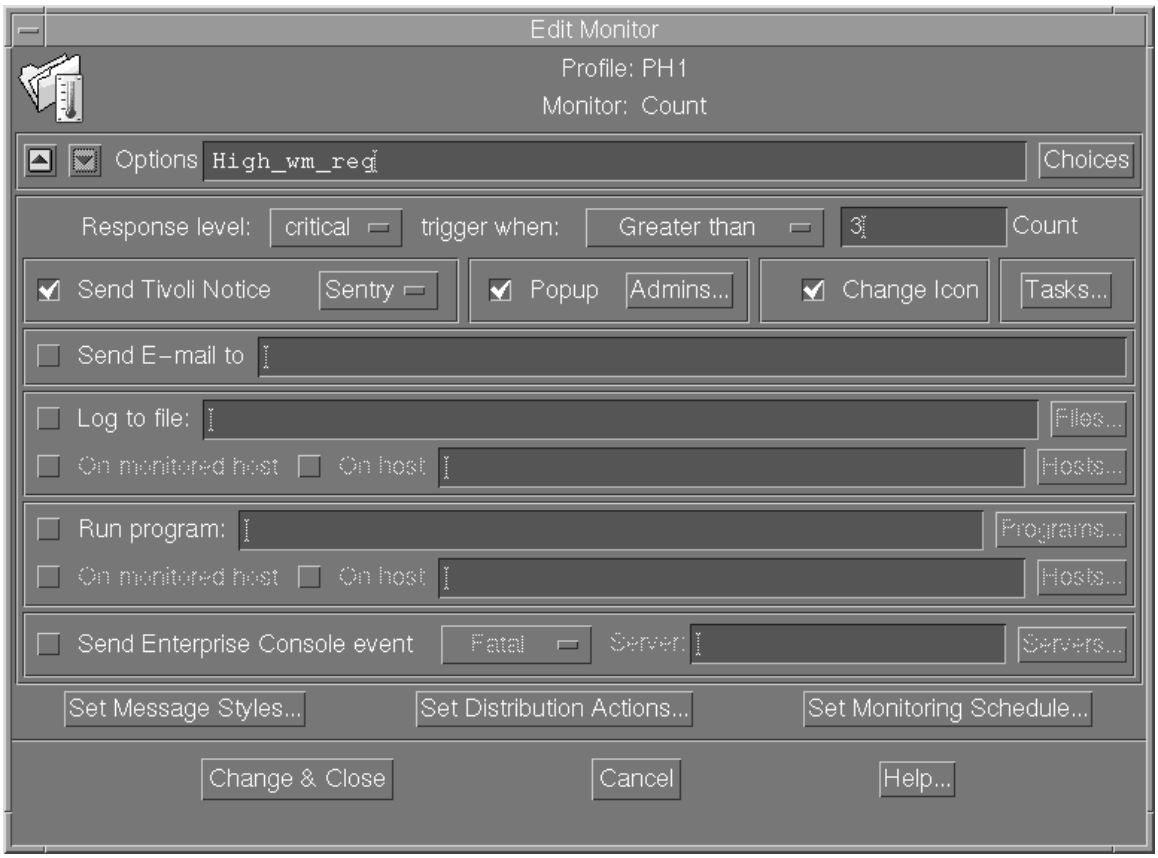


Figure 126. snapm1 - Monitor Properties

We want to know when the High_wm_reg value becomes critical and select Response level **critical**, trigger when **Greater than**, and enter 3 in the Count box. We select **Send Tivoli Notice** to get information in the Notices Board, **Popup** to get a pop-up window every time the monitor exceeds the threshold, and **Change Icon** to have the icon on an indicator collection reflect changes. If desired, change the default monitoring period of 1 hour. Select **Change & Close** and the window in Figure 127 on page 138 is displayed, showing that we have one monitor, Count (dbmsadm, High_wm_reg), in our profile PH1.

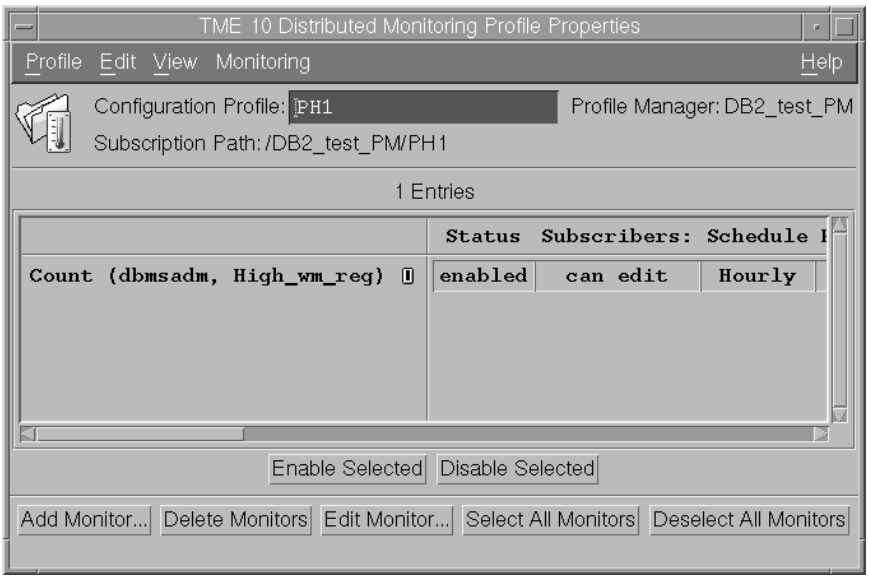


Figure 127. snapm1 - Monitors

We select **Profile** from the menu bar and then **Save** from the pull-down menu. Then we select **Profile** from the menu bar again and then **Exit** from the pull-down menu.

Figure 128. snapm1 - Distribute Profile

We distribute the profile PH1 to the subscriber rs600019, which is our TMR server and also the node on which we run DB2. We can do this, for example, by selecting the **PH1** icon with the right mouse button pressed and dragging it onto the **rs600019** icon.

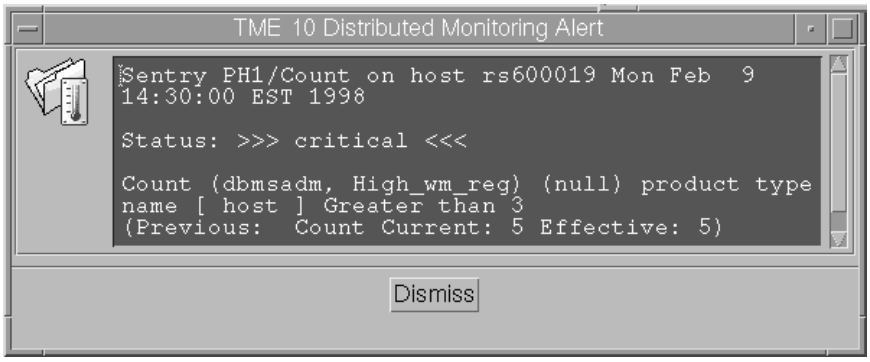


Figure 129. Pop-Up for snapm1 Counters

When the monitor begins, the pop-up in Figure 129 is the result. It shows that we have a critical status on rs600019 for the High_wm_reg parameter. The current value is 5, which is above the threshold of 3 we set in Figure 126 on page 137.

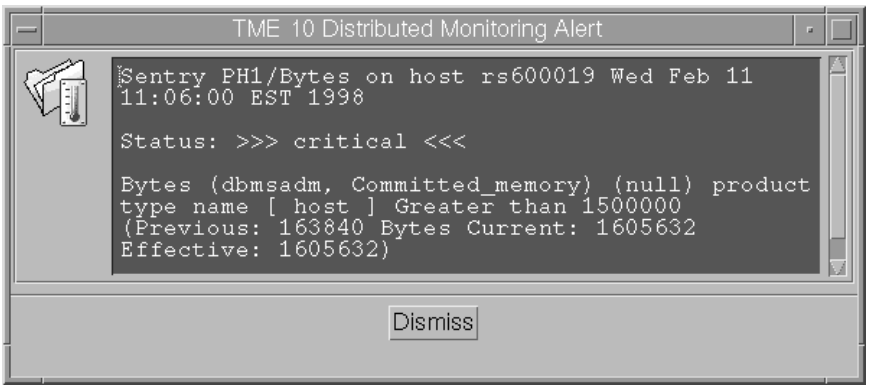


Figure 130. Pop-Up for snapm2 Memory

Similar to the Counter monitor, the Bytes monitor shows the result as in Figure 130. The alert pop-up shows that we have a critical status on committed private memory and that we have exceeded 1500000 bytes.

Note

You must have the same number of arguments in your implementation shell script or C program as you want to be displayed in the alert. In our case we have DBM admin ID as an argument and must have Committed Memory as well, even though our shell script doesn't use the second argument. If we only specify the first argument in the snapm.csl for the Memory monitor, our alert would just show Bytes (dbmsadm) instead of as now Bytes (dbmsadm, Committed_memory).

3.12.6 Database Monitor

The previous monitor was an RDBMS-specific monitor that can be used to monitor general parameters of the RDBMS. We are now going to add another monitor that can be used to monitor specific databases within the RDBMS.

This monitoring collection consists of two monitors, one monitoring counters and unit_of_work and the other one monitoring database status. The value for unit_of_work is defined as the sum of Commit Statements Attempted, Internal Commits, Rollback Statements Attempted, and Internal Rollbacks. This calculation is done in the snapd2.sh script, which is listed in Figure 135 on page 146.

As mentioned before, the Snapshot Monitor does not always return values that we can do calculation on; we need to test first if the data is available or not. This is done in both snapd1.sh and snapd2.sh, where we test for a specific message, SQL1611W. If we get this message, we will return the value 0 for the Counters monitor and the value Unknown for the Status monitor.

With the Counters monitor for databases, we decide just to look at a few counters; the full listing of the different counters can be seen in Figure 131 on page 141. In the example we show the database used by Tivoli Inventory.

```
              Database Snapshot

Database name                               = TIV_INV
Database path                               = /home/dbmsadm/database/dbmsadm/SQL00001/
Input database alias                        = TIV_INV
Database status                             = Active
Locks held currently                        = 9
Lock waits                                  = 0
Time database waited on locks (ms)          = 0
Lock list memory in use (Bytes)             = 1116
Deadlocks detected                          = 0
Lock escalations                            = 0
Exclusive lock escalations                  = 0
Current applications waiting on locks       = 0
Lock Timeouts                               = 0
Total sort heap allocated                   = 0
Total sorts                                 = Not Collected
Total sort time (ms)                        = Not Collected
Sort overflows                              = Not Collected
Active sorts                                = 0
Buffer pool data logical reads              = 242
Buffer pool data physical reads             = 46
Asynchronous pool data page reads           = 0
Buffer pool data writes                     = 0
Asynchronous pool data page writes          = 0
Buffer pool index logical reads             = 194
Buffer pool index physical reads            = 34
Buffer pool index writes                    = 0
Asynchronous pool index page writes         = 0
Total buffer pool read time (ms)            = 0
Total buffer pool write time (ms)           = 0
Total elapsed asynchronous read time        = 0
Total elapsed asynchronous write time       = 0
Asynchronous read requests                  = 0
LSN Gap cleaner triggers                    = 0
Dirty page steal cleaner triggers           = 0
Dirty page threshold cleaner triggers       = 0
Direct reads                                = 412
Direct writes                               = 136
Direct read requests                        = 37
Direct write requests                       = 2
Direct reads elapsed time (ms)              = 2819038593
Direct write elapsed time (ms)              = 0
Database files closed                       = 0
Commit statements attempted                 = 10
Rollback statements attempted               = 5
Dynamic statements attempted                = 1102
Static statements attempted                 = 15
Failed statement operations                 = 1
Select SQL statements executed              = 11
Update/Insert/Delete statements executed    = 0
DDL statements executed                     = 0
```

Figure 131 (Part 1 of 2). Output from db2 get snapshot for database on tiv_inv Command

```
Internal automatic rebinds                 = 0
Internal rows deleted                      = 0
Internal rows inserted                     = 0
Internal rows updated                      = 0
Internal commits                           = 11
Internal rollbacks                         = 0
Internal rollbacks due to deadlock         = 0
Rows deleted                               = 0
Rows inserted                              = 0
Rows updated                               = 0
Rows selected                              = 1057
Binds/precompiles attempted                = 0
First database connect timestamp           = 02-09-2008 11:03:25.999999
Last reset timestamp                       =
Last backup timestamp                      =
Snapshot timestamp                         = 02-09-2008 11:03:25.999999
High water mark for connections            = 2
High water mark for database heap          = 448127
Application connects                       = 6
Applications connected currently           = 1
Appls. executing in db manager currently   = 0
Maximum secondary log space used (Bytes)   = 0
Maximum total log space used (Bytes)       = 0
Secondary logs allocated currently         = 0
Log pages read                             = 0
Log pages written                          = 0
Package cache lookups                      = 23
Package cache inserts                      = 11
Catalog cache lookups                      = 57
Catalog cache inserts                      = 13
Catalog cache overflows                    = 0
Catalog cache heap full                    = 0
```

Figure 131 (Part 2 of 2). Output from db2 get snapshot for database on tiv_inv Command

The following files are the elements of the monitoring collection, DB2_Database:

- snapd.msg

```
$ key=snapd_Descr
1 Counters
$ key=snapd_Help
2 This Monitor returns the current value on a selected counter from the output
of the DB2 Snapshot Monitor. The unit of work is calculated as commit statements
attempted + internal commits + rollback statements + internal rollbacks. This is
according to the manual DB2 Database System Monitor Guide and Reference.
$ key=snapd_val_Descr
3 Count
$ key=snapd_list
4 Options
$ key=snapd_Locks_held
5 Locks held
$ key=snapd_Lock_waits
6 Lock waits
$ key=snapd_Deadlocks
7 Deadlocks
$ key=snapd_Unit_of_work
8 Unit of work calculated
$ key=generic_help
9 The monitor extracts counters and other information from the snapshot output.
For more information please use the DB2 Database System Monitor Guide and Reference.
$ key=ButtonLabelChoice
10 Choices
$ key=snapd2_val_Descr
11 Active/Quiesce-pend/Quiesced
$ key=snapd2_Descr
12 Status
$ key=snapd2_Help
13 This Monitor returns the status of the database in question. The status can be
Active, Quiesce-pending, or Quiesced. For information on the different values please
refer to the DB2 Database System Monitor Guide and Reference.
$ key=snapd_DBM_id
14 DBM admin id
$ key=snapd_Database_id
15 Database id
```

Figure 132. snapd.msg Listing

Note

Do not use a new line when you add text for a variable. If you force a new line, you will only see the first line of text when you select help in Tivoli Distributed Monitoring.

- snapd.csl

```
#include "Sentry2_0.dsl"
#include "snapd.dsl"

Collection "DB2_Database" {
    CodeID = "$Id:snapd.csl,v 1.0$";
    Version = "1.0";
    Require = ">2.0.2";
    HelpMessage = (snapd_generic_help);
    EventBaseClass = "Sample_Sentry_Monitors";
    NoticeGroup = "Sentry";

#include "operators.csl"
#include "choicelists.csl"
#include "formats.csl"

    ChoiceList DiffOptions {
      ButtonLabel = (snapd_ButtonLabelChoice);
      {
         { (snapd_snapd_Locks_held) "Locks_held" }
         { (snapd_snapd_Lock_waits) "Lock_waits" }
         { (snapd_snapd_Deadlocks) "Deadlocks" }
         { (snapd_snapd_Unit_of_work) "UOW" }
       };
    };
    Monitor Counters Numeric Group numeric {
      Description = (snapd_snapd_Descr);
      ValueDescription = (snapd_snapd_val_Descr);
      HelpMessage = (snapd_snapd_Help);
      Argument (snapd_snapd_DBM_id)
        DefaultValue "dbmsadm";
      Argument (snapd_snapd_Database_id)
        DefaultValue "database";
      Argument (snapd_snapd_list)
        RestrictedChoice "DiffOptions"
        DefaultValue "Locks_held";
      Implementation (aix3-r2, aix4-r1)
      Shell("/bin/ksh", "-c", Command, "snapd1")
      Import "snapd1.sh"
     ;
    };
    Monitor Status String Group string {
      Description = (snapd_snapd2_Descr);
      ValueDescription = (snapd_snapd2_val_Descr);
      HelpMessage = (snapd_snapd2_Help);
      Argument (snapd_snapd_DBM_id)
        DefaultValue "dbmsadm";
      Argument (snapd_snapd_Database_id)
        DefaultValue "database";
      Implementation (aix3-r2, aix4-r1)
      Shell("/bin/ksh", "-c", Command, "snapd2")
      Import "snapd2.sh"
     ;
    };
}
```

Figure 133. snapd.csl Listing

As can be seen in the snapd.csl listing in Figure 133 there are two defined monitors: Counters monitor, which is a numeric monitor that calls snapd1.sh and the Status monitor, which is a string monitor that calls snapd2.sh.

- snapd1.sh

```
#!/bin/ksh
#
# snapd1.sh
# sample monitor for DB2 for AIX
# that will analyze the output from snapshot,
# regarding different counters for a specific
# database.
#
# Peter Holm, PHOLM@SE.IBM.COM
# ITSO Raleigh 1998
#
#

if [ "$1" = "" ]
then
  print "Specify DB admin's ID as first parameter."
  exit 1
fi
#
# get snapshot
#
su - $1 -c "db2 get snapshot for database on" $2 >/tmp/snapd

# First check if we have no data returned, then just leave

m=$(grep "SQL1611W" /tmp/snapd| cut -d ' ' -f1)
if [ "$m" = "SQL1611W" ]
then
  echo 0
  exit 0
fi

case $3 in
  "Locks_held" )   m=$(grep "Locks held" /tmp/snapd | cut -d'=' -f2);;
  "Lock_waits" )   m=$(grep "Lock waits" /tmp/snapd | cut -d'=' -f2);;
  "Deadlocks" )    m=$(grep "Deadlocks" /tmp/snapd | cut -d'=' -f2);;
  "UOW" )          m1=$(grep "Commit statements attempted" /tmp/snapd | cut -d'=' -f2)
                   m2=$(grep "Internal commits" /tmp/snapd | cut -d'=' -f2)
                   m3=$(grep "Rollback statements attempted" /tmp/snapd | cut -d'=' -f2)
                   m4=$(grep "Internal rollbacks   " /tmp/snapd | cut -d'=' -f2)
                   m=$(($m1+$m2+$m3+$m4));;
esac

echo $m
exit 0
```

Figure 134. snapd1.sh Listing

The snapd1.sh shell script has three arguments: DB admin ID, database alias, and counter name. There is a test on message SQL1611W, which we get if there is no data available. In that case we return the value 0.

- snapd2.sh

```
#!/bin/ksh
#
# snapd2.sh
# sample monitor for DB2 for AIX
# that will analyze the output from snapshot,
# regarding status of a specific
# database.
#
# Peter Holm, PHOLM@SE.IBM.COM
# ITSO Raleigh 1998
#
#

if ffl "$1" = "" “
then
  print "Specify DB admin's ID as first parameter."
  exit 1
fi
#
# get snapshot
#
su - $1 -c "db2 get snapshot for database on" $2 >/tmp/snapd

# First check if we have no data returned, then just leave

m=$(grep "SQL1611W" /tmp/snapd| cut -d ' ' -f1)
if ffl "$m" = "SQL1611W" “
then
  echo Unknown
  exit 0
fi

m=$(grep "Database status" /tmp/snapd | cut -d'=' -f2)
echo $m

exit 0
```

Figure 135. snapd2.sh Listing

The Status monitor has two arguments: DBM admin ID and database alias.

To create the monitoring collection we now issue the following commands:

1. `gencmsg snapd.msg`

   to compile the message file for snapd.

2. `mcsl -P ./cpp -x ./snapd.col snapd.csl -lang-c++ nostdinc undef`

   to compile the snapd monitoring collection.

3. `mcsl -iR snapd.col`

   to install the snapd monitoring collection.

To make your changes effective you'll need to recycle oserv; this is done with:

```
odadmin reexec all
```

3.12.7 Running the Database Monitor

After doing the necessary compilations, we now have another ready monitoring collection. As we don't always get any useful data with the Snapshot Monitor, we need a way to generate load on the system. We will use the queries that come with Tivoli Inventory to access the database and create a sufficient load for our purposes. So while the monitor runs we will do some queries on hardware and software data.

We start with defining a profile manager and a Tivoli Distributed Monitoring profile. We open the profile and click on **Add Monitor...** and the following window is displayed:

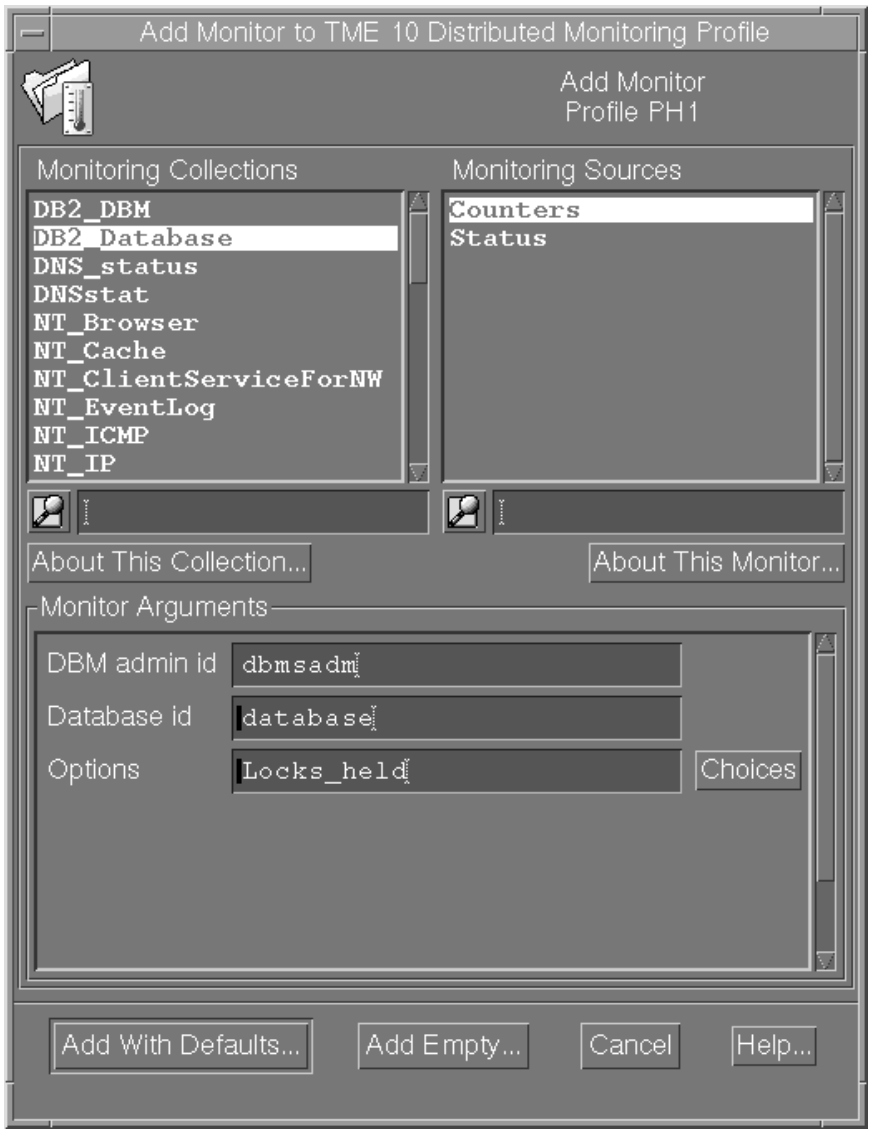


Figure 136. snapd1 - Add Monitor

We first select the **DB2_Database** monitoring collection and then we select the **Counters** monitoring source, and the Monitoring Arguments show up in Figure 136. We want to monitor the unit_of_work, and click on **Choices**.

Figure 137. snapd1 - Options

From the list we select the desired value, **Unit of work calculated** in our example. Then we select **Set & Close**.

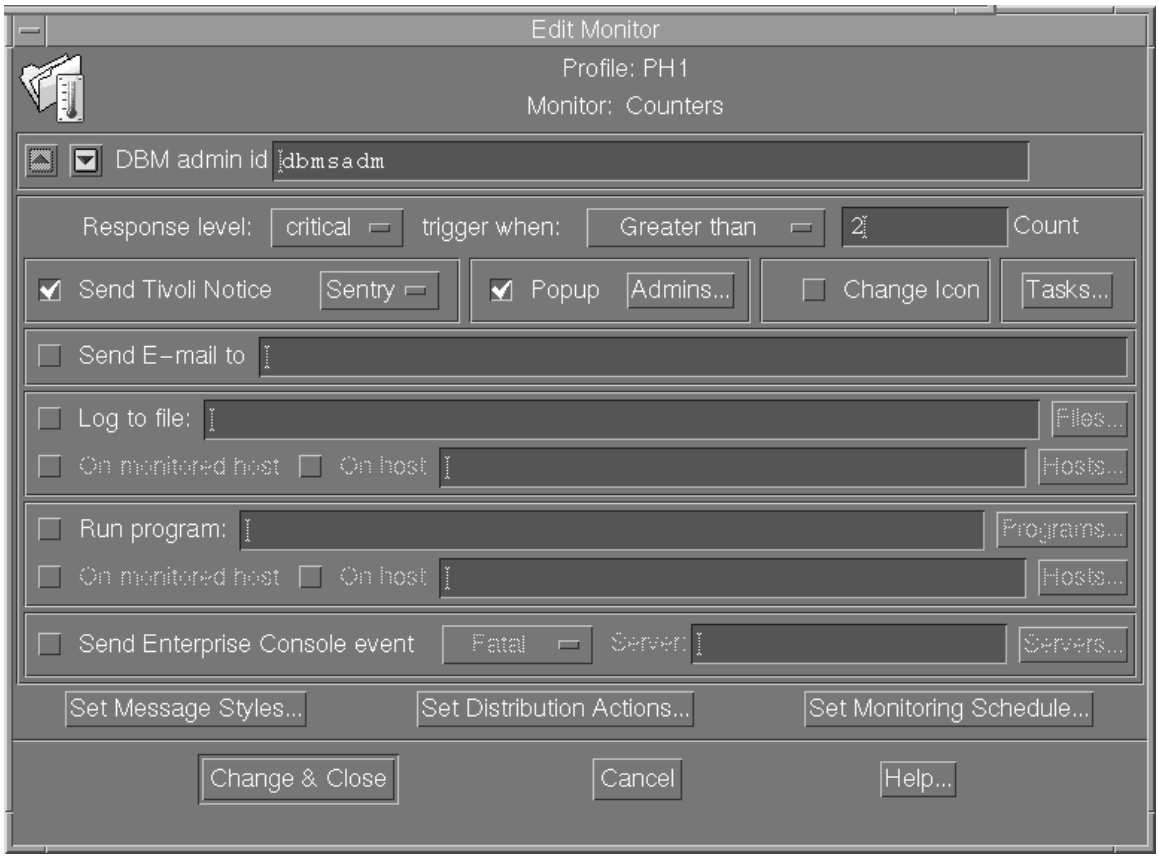


Figure 138. snapd1 - Monitor Properties

We want to know when the unit_of_work becomes critical and select Response level **critical**, trigger when **Greater than**, and enter 2 as count. We select **Popup** to get a pop-up message when the threshold is exceeded, **Send Tivoli Notice** to get entries in the notice board, and **Change Icon** to have the indicator collection icon reflect changes.

If desired, change the default monitoring period of 1 hour. Select **Change & Close**.

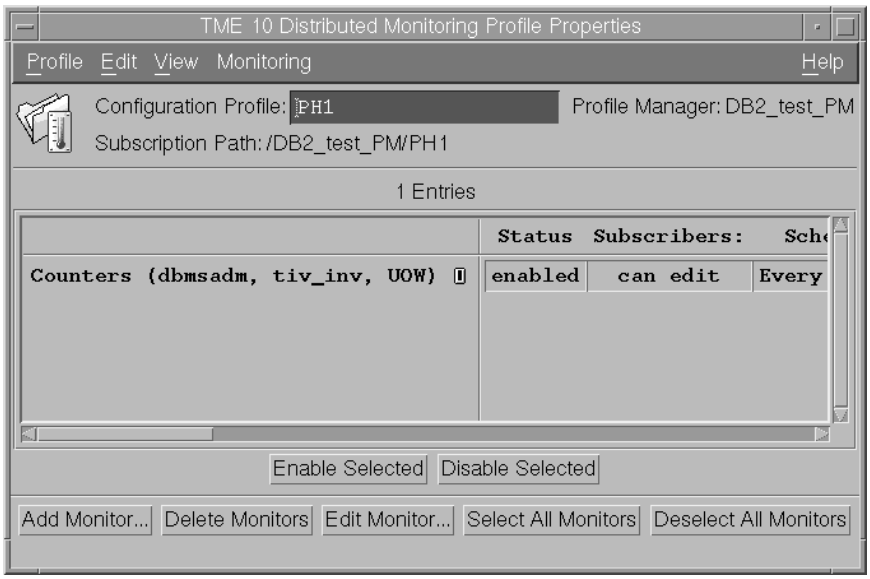


Figure 139. snapd1 - Monitors

We select **Profile** from the menu bar and then **Save** from the pull-down menu. Then we select **Profile** from the menu bar again and then **Exit** from the pull-down menu. Then we distribute the profile PH1 to the subscriber rs600019, which is our TMR server and also the node on which we run DB2.

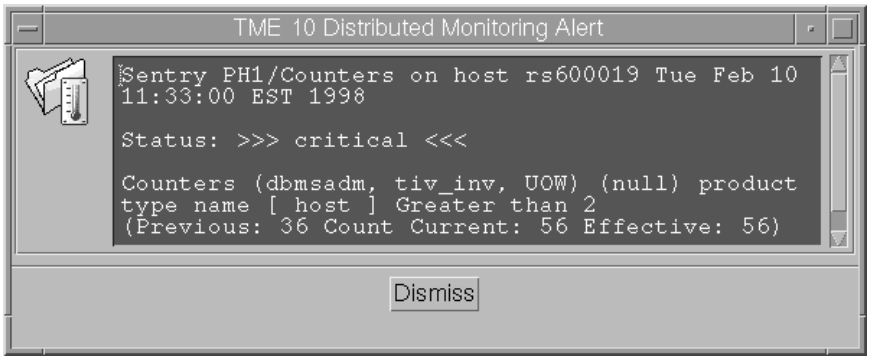


Figure 140. Pop-up - snapd1 Counters

When the monitor begins the pop-up in Figure 140 is the result. We have critical status, where the value for UOW is 56. The previous is 36 and our threshold is 2.

Similar to the Counter monitor, the Status monitor shows the result as in Figure 141 on page 152 with the Unknown status and in Figure 142 on page 152 the Active status.

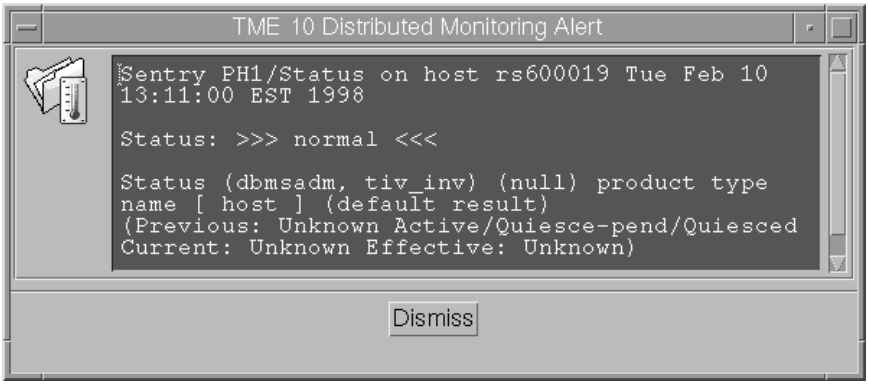


Figure 141. Pop-Up snapd2 Status Unknown

We defined the Unknown status in our snapd2.sh shell script as the result of no data available from the snapshot command.

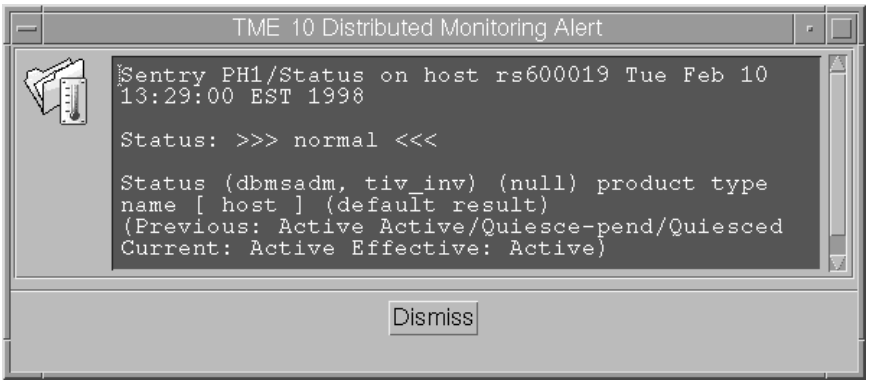


Figure 142. Pop-Up snapd2 Status Active

This alert is triggered with response level always just so that we see the result.

3.12.8 DB2 Monitoring Summary

We now have four monitors in two monitoring collections for DB2 monitoring: two that monitor the overall behavior of DB2 and two that monitor a specific database. These monitors should provide a good base for further customization of Tivoli Distributed Monitoring with regard to DB2.

3.13 Monitoring File Systems

The reason for creating this monitoring collection is twofold. First, we want to show that it is possible to monitor multiple resources with Tivoli Distributed Monitoring although Tivoli Distributed Monitoring itself has a limit on this, in that it can only monitor one resource per monitor. Secondly, we know that monitoring of file systems on file servers is a main headache for customers, who may easily have 150-200 file systems per server. And there is no way they want to implement monitoring of such a server with one monitor per resource!

Our solution to this problem is to have a program locally monitoring specific file systems against thresholds, and in the event of a threshold exceeded there will be a Tivoli Distributed Monitoring alert sent and a TEC event posted to the TEC server. The TEC event is sent by means of postemsg.

In the overview in Figure 143 on page 153, you can see how the fsmon program sends the TEC events. Another solution could have been to have Tivoli Distributed Monitoring send the TEC event, but in such a solution we would only get one event per node, and not one event for each exceeded file system.

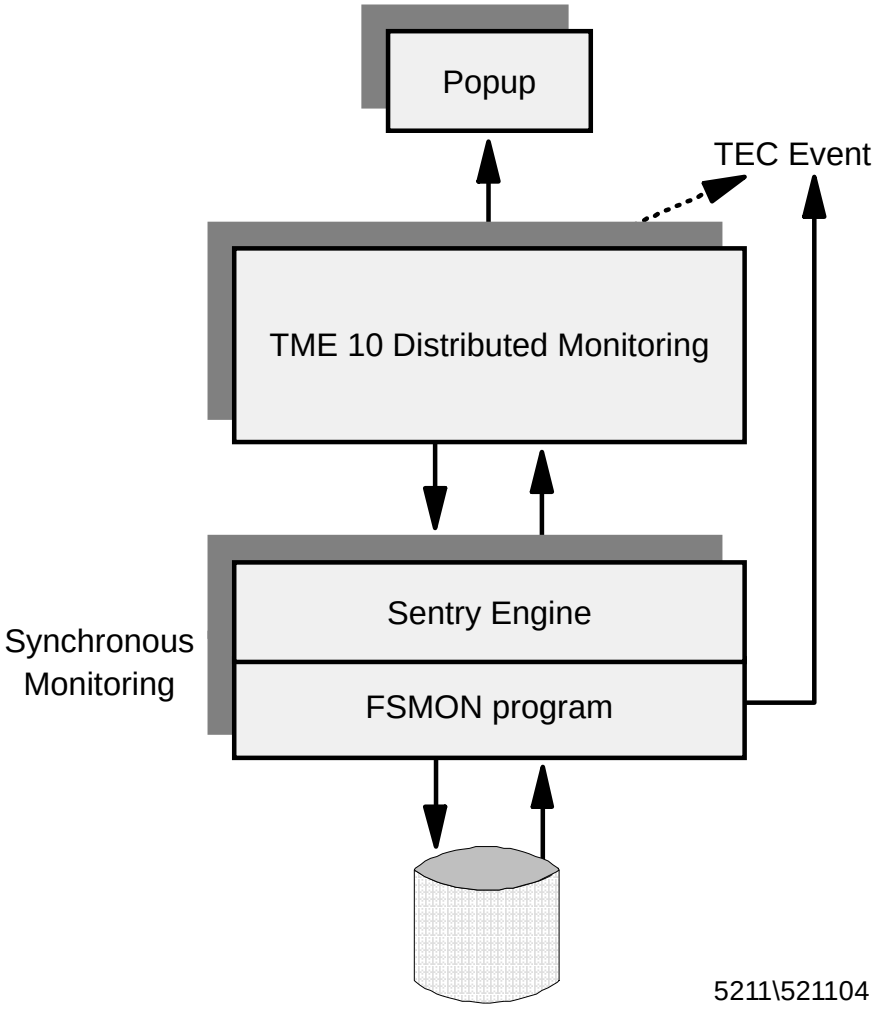


Figure 143. View of Event Flow

The strength of the monitor program is that it will accept generic file system names, for instance /usr$ will monitor all file systems starting with /usr. So, if you have a good naming convention, you don't need too many entries to cover your file systems. The program has been tested on AIX, SunOS, Solaris, and HP-UX.

Note

There is an undocumented limitation in Tivoli Distributed Monitoring that only eight arguments will be displayed in the Monitoring Arguments when adding a monitor. The MCSL compilation does not indicate any error if you specify more than eight arguments.

Note

The reason for using $ instead of *, which is the commonly used character for generic definitions, is that the Sun operating systems strip off the asterisk when passing arguments to the called program.

In Figure 144 we show the output from the df -k command on AIX 4.2.

```
root@rs600019:/# df -k
Filesystem    1024-blocks      Free %Used    Iused %Iused Mounted on
/dev/hd4            28672     16632   42%     2173    16% /
/dev/hd2           491520     56456   89%    16868    14% /usr
/dev/hd9var        143360    118572   18%      916     3% /var
/dev/hd3            24576     16692   33%      235     4% /tmp
/dev/hd1            40960     15684   62%      429     5% /home
/dev/lv00          253952     95380   63%      369     1% /usr/local/sybase
/dev/lv01          307200    118564   62%     7720    10% /usr/local/Tivoli
/dev/lv02          102400     74192   28%      144     1% /var/spool/Tivoli
/dev/lv03          102400     33856   67%      117     1% /home/dbmsadm/database
```

Figure 144. Output from df -k Command on AIX

In Figure 145 on page 155 we show the output from the df -lk command on HP-UX 10. As you can see it differs totally from the previous AIX output.

```
/gcc                    (/dev/vg00/lvo20       ) :    89702 total allocated Kb
                                                      74076 free allocated Kb
                                                      15626 used allocated Kb
                                                         17 % allocation used
/home                   (/dev/vg00/lvol3       ) :    17874 total allocated Kb
                                                      16686 free allocated Kb
                                                       1188 used allocated Kb
                                                          6 % allocation used
/opt                    (/dev/vg00/lvol4       ) :    93273 total allocated Kb
                                                      65706 free allocated Kb
                                                      27567 used allocated Kb
                                                         29 % allocation used
/tmp                    (/dev/vg00/lvol5       ) :    21503 total allocated Kb
                                                      21371 free allocated Kb
                                                        132 used allocated Kb
                                                          0 % allocation used
/usr/local/Tivoli       (/dev/vg00/lvol9       ) :   179442 total allocated Kb
                                                       40248 free allocated Kb
                                                      139194 used allocated Kb
                                                         77 % allocation used
/usr                    (/dev/vg00/lvol6       ) :   297983 total allocated Kb
                                                      100364 free allocated Kb
                                                      197619 used allocated Kb
                                                         66 % allocation used
/var/spool/Tivoli       (/dev/vg00/lvol10      ) :    17874 total allocated Kb
                                                      17779 free allocated Kb
                                                         95 used allocated Kb
                                                          0 % allocation used
/var                    (/dev/vg00/lvol7       ) :    64588 total allocated Kb
                                                      48958 free allocated Kb
                                                      15630 used allocated Kb
                                                         24 % allocation used
/                       (/dev/vg00/lvol1       ) :    43046 total allocated Kb
                                                      13021 free allocated Kb
                                                      30025 used allocated Kb
                                                         69 % allocation used
```

Figure 145. Output from df -lk Command on HP-UX

This monitoring collection implements the monitor in C. We call the program fsmon and it covers all three supported platforms. It could be split into three different programs, one for each platform, but we decided to place everything in one program.

Running the program fsmon against the file systems on rs600019 with a file system parameter of /usr$: 60 as displayed in Figure 144 on page 154 would result in one Tivoli Distributed Monitoring alert for /usr, /usr/local/sybase and /usr/local/Tivoli, and three TEC events, one for each of the file systems being exceeded. In the Tivoli Distributed Monitoring alert, the different file systems will be divided by a vertical bar. If there is no file system exceeding the threshold, an OK is returned. The TEC event will have more information than the Tivoli Distributed Monitoring alert message.

Note

There is a difference between shell scripts and C programs in that the shell scripts can be imported into the monitoring collection and can be distributed together with the monitor. The C programs need to be compiled for each platform and then put into a directory in the $PATH on each node.

These are the different steps involved in using the fsmon monitoring collection:

1. Define file systems to be monitored together with a threshold.

   Generic file system names can be used in the form /usr$, /usr/l$ or exact file system names.

2. Distribute the monitor.

3. If a threshold is exceeded a Tivoli Distributed Monitoring alert will be generated.

   Depending on your definitions it might be a pop-up, notice, or icon change of an indicator collection.

   The fsmon program will print the following string when a threshold is exceeded:

   ```
   Node=nodename_fs=filesystem_usage=nn
   ```

   If more than one file system is above its threshold, the string will be concatenated.

4. Together with the Tivoli Distributed Monitoring alert the fsmon program will generate a TEC event.

   One TEC event for each exceeded threshold will be generated.

3.13.1 The Monitor Definitions

The monitoring collection consists of one message file fsmond.msg and a csl file fsmond.csl. As can be seen in Figure 146 on page 157 the message defines fields for the different TEC severities, TEC server, and file system with corresponding thresholds.

```
$ key=fsmond_Descr
1 Filesystem usage
$ key=fsmond_Help
2 This Monitor will measure the percentage space used for specific filesystems.
Up to 3 filesystems can be specified together with a threshold value for each filesystem.
This threshold is used to decide if the filesystem usage is critical or not. If a threshold
is exceeded for a filesystem, the filesystems that are found are returned by the monitor in
a string as follows: hostname_filesystem_usage. All detailed information will be sent to the
TEC server using postemsg command, which is part of the TEC. If TEC server is not specified
posting of event will not be done. The postemsg binary must exist on each system, where this
monitor will run. For further information on how to install this monitor please refer to the
Redbook SG24-5211, Creating Custom Monitoring Collections.
$ key=fsmond_val_Descr
3 Filesystems
$ key=fsmond_tecserver
4 TEC server
$ key=fsmond_severity
5 Severity
$ key=fsmond_Fatal
6 FATAL
$ key=fsmond_Critical
7 CRITICAL
$ key=fsmond_Minor
8 MINOR
$ key=fsmond_Warning
9 WARNING
$ key=fsmond_Harmless
10 HARMLESS
$ key=ButtonLabelChoice
11 TEC status
$ key=fsmond_fs1
12 Filesystem1
$ key=fsmond_fst1
13 Filesystem1 threshold
$ key=fsmond_fs2
14 Filesystem2
$ key=fsmond_fst2
15 Filesystem2 threshold
$ key=fsmond_fs3
16 Filesystem3
$ key=fsmond_fst3
17 Filesystem3 threshold
$ key=generic_help
18 The monitor will monitor filesystem space usage. For more information please use the
Redbook SG24-5211, Creating Custom Monitoring Collections.
```

Figure 146. Listing of fsmond.msg

The csl file listed in Figure 147 on page 158 defines a list button for the TEC severities. There is only one monitor in this monitoring collection, namely Filesystems, and it is a string monitor. The implementation is defined for three platforms, one different loadmodule for each platform. You don't need different loadmodule names, but it easier to keep track of this way.

```
#include "Sentry2_0.dsl"
#include "fsmond.dsl"

Collection "FSMon" {
    CodeID = "$Id:fsmond.csl,v 1.0$";
    Version = "1.0";
    Require = ">2.0.2";
    HelpMessage = (fsmond_generic_help);
    EventBaseClass = "Sample_Sentry_Monitors";
    NoticeGroup = "Sentry";

#include "operators.csl"
#include "formats.csl"
#include "choicelists.csl"

    ChoiceList Severity {
      ButtonLabel = (fsmond_ButtonLabelChoice);
      {
         { (fsmond_fsmond_Harmless) "HARMLESS" }
         { (fsmond_fsmond_Warning) "WARNING" }
         { (fsmond_fsmond_Minor) "MINOR" }
         { (fsmond_fsmond_Critical) "CRITICAL" }
         { (fsmond_fsmond_Fatal) "FATAL" }
      };
    };

     Monitor Filesystems String Group string {
      Description = (fsmond_fsmond_Descr);
      ValueDescription = (fsmond_fsmond_val_Descr);
      HelpMessage = (fsmond_fsmond_Help);
      Argument (fsmond_fsmond_tecserver)
        DefaultValue "rs600019";
      Argument (fsmond_fsmond_severity)
        RestrictedChoice "Severity"
        DefaultValue "CRITICAL";
      Argument (fsmond_fsmond_fs1)
        DefaultValue "";
      Argument (fsmond_fsmond_fst1)
        DefaultValue "";
      Argument (fsmond_fsmond_fs2)
        DefaultValue "";
      Argument (fsmond_fsmond_fst2)
        DefaultValue "";
      Argument (fsmond_fsmond_fs3)
        DefaultValue "";
      Argument (fsmond_fsmond_fst3)
        DefaultValue "";
      Implementation (aix3-r2, aix4-r1)
      Shell("fsmonaix");
      Implementation (sunos4, solaris)
      Shell("fsmonsun");
      Implementation (hpux10)
      Shell("fsmonhp");
  };
}
```

Figure 147. Listing of fsmond.csl

3.13.2 The FSMON Program

The program supports, as mentioned before, AIX, SunOS, Solaris, and HP-UX 10. It will check what platform it is running on and then execute the corresponding code. The program can easily be extended to support other platforms, as there are two sections for each platform, namely:

- validate_usage_xxx

  This section checks whether we have an exact match or generic match.

- parse_df_record_xxx

  This section just extracts the information from the result of the df command. For HP-UX this section doesn't exists as the output from the df command on HP-UX differs a lot, with multiple lines per entry.

The source listing of fsmon.c is shown in Figure 148 on page 160. We will discuss some parts of the program onward, just so you will understand what it is doing.

We decide to compile the fsmon.c program on the different platforms and keep the loadmodules on the TMR server as a start. To be able to differentiate between the different loadmodules, we name them fsmonaix, fsmonsun, and fsmonhp.

The compilers used on AIX are the standard C compilers. On Sun and HP-UX, we use the GNU C compiler available as freeware on the Internet. To compile the source code we use:

```
cc fsmon.c -o fsmonaix -w
```

or

```
gcc fsmon.c -o fsmonsun -w
```

```
/*********************************************************************/
/*                                                                   */
/* Author Peter Holm, PSS, IBM Nordic                                */
/*           e-mail: pholm@se.ibm.com                                */
/*                                                                   */
/*  (C) Copyright IBM Corp. 1998                                     */
/*                                                                   */
/*********************************************************************/
/* fsmon                        Last update: 1998-02-14              */
/*                                                                   */
/* Description:                                                      */
/*     This program will check the filesystem usage.                 */
/*     It will run on Sun Solaris, HP-UX 10, AIX 3.2.5,              */
/*     and AIX 4.x                                                   */
/*     The program will find out the platform it's running on,       */
/*     but it needs to be compiled for each platform as the binaries */
/*     differ.                                                       */
/*                                                                   */
/*     Input parameters: filesystemname + usage threshold(%) in pairs. */
/*                       Up to 200 filesystems are supported.        */
/*                       Generic values for filesystem name are ok.  */
/*                                                                   */
/*                       asterisk cannot be used for Sun, as that    */
/*                       character is stripped when passing          */
/*                       argument to program. That's why we use $.   */
/*                                                                   */
/*                                                                   */
/*     Output values:    string = OK  - if all filesystems are ok.   */
/*                       string = node_filesystem_current usage  -   */
/*                                for the first filesystem found that */
/*                                is above threshold.                */
/*                                                                   */
/*     T/EC interface:   An event is sent to T/EC server, using the  */
/*                       Framework wpostemsg for each of the         */
/*                       filesystems that exceed threshold, not just */
/*                       the first one.                              */
/*                                                                   */
/*                                                                   */
/* Compile:                                                          */
/*     cc fsmon.c -o fsmonaix -w                                     */
/*                                                                   */
/* Changed:                                                          */
/*     By  Date    Description                                       */
/*     --  ------  ----------------                                  */
/*                                                                   */
/*********************************************************************/
```

Figure 148. (Part 1 of 12). Source Listing of fsmon.c

This program supports up to 200 file systems, but as stated before there is currently a limitation in Tivoli Distributed Monitoring that only eight arguments can be passed to a called program.

```
/* ************************************************** */
/* include files                                      */
/* ************************************************** */
#include <stdio.h>
#include <errno.h>
#include <time.h>
#include <string.h>
#include <stdlib.h>

/* ************************************************** */
/* declare constants                                  */
/* ************************************************** */
#define BUFSIZE 250         /* used in print files */
#define MAXFSLINES 200   /* max number of filesystems we can monitor */
#define PATHMAX 50
/* ************************************************** */
/* declare variables                                  */
/* ************************************************** */
int i, j, current_lines;
int first_occ, prcntused;
int errno, rc;
int fsu[MAXFSLINES];
char sevflag[8];
char mounted[PATHMAX];
char TECSERVER[50];
char prog[8]="fsmon";
char *fs[MAXFSLINES];
char *ptr_fs;
char alertmsg[BUFSIZE];
char eventmsg[BUFSIZE];
char hostname[16];
char opsystem[16];
char rel[16];
char version[16];

/* ************************************************** */
/* declare functions                                  */
/* ************************************************** */
int get_hostdata(void);
int validate_usage_sun(void);
int validate_usage_aix(void);
int validate_usage_hp(void);
void set_Tivoli_env(void);
void send_TEC(char *mounted, int *size, int *used, int *free);
void first_occur(char *mounted, int *used);
```

Figure 149. (Part 2 of 12). Source Listing of fsmon.c

```
/* ************************************************** */
/* main routine                                       */
/* ************************************************** */
main(argc,argv)
int  argc;
char *argv[];
{
if (argc < 4)
   {
    fprintf(stderr, "Wrong number of parameters\n");
    exit(1);
   }
strcpy(TECSERVER,argv[1]);
strcpy(sevflag,argv[2]);
i=3;
j=0;
   while (i<argc)
     {
      fs[j]=argv[i];
      rc=sscanf(argv[i+1], "%d", &fsu[j]);
      i=i+2;
      j=j+1;
     }
   set_Tivoli_env();
   if (get_hostdata() !=0)
     {
      fprintf(stderr,"Error getting hostdata\n");
      exit(-1);
     }
   current_lines=j;
   first_occ=0;
   /* Validate usage                    */
   if (strncmp(opsystem, "SunOS", 5) == 0)
      validate_usage_sun();
   if (strncmp(opsystem, "AIX", 3) == 0)
      validate_usage_aix();
   if (strncmp(opsystem, "HP-UX", 5) == 0)
      validate_usage_hp();
   /* Now check if we have any threshold exceeds, if not print "OK" */
   if (first_occ == 0)
     {
       printf("OK");
     }
     else
     {
       printf(alertmsg);
     }
exit(0);
}
```

Figure 150. (Part 3 of 12). Source Listing of fsmon.c

This is the main routine that reads the arguments, tests for validity, calls get_hostdata to figure out the current platform, and calls the platform-specific routine.

Before exiting, a test is made if we had any threshold exceeded or not.

```
/* ************************************************ */
/* Validate usage subroutine for SunOS              */
/* ************************************************ */
int validate_usage_sun()
{

     FILE *wfile;              /* used for df commands */
     int free, size, used;
     char mounted[PATHMAX];
     int rc=0, rx=0;
     char cmd[30], wline1[BUFSIZE];

     sprintf(cmd, "df -lk");  /* df -lk displays only local fs */

     if ((wfile=popen(cmd, "r")) == NULL)
        {
          fprintf(stderr,"Error opening pipe for df for SunOS\n");
          exit(-1);
        }

     if (fgets(wline1, sizeof(wline1), wfile) == NULL)
        {
          fprintf(stderr,"Error parsing first header record from pipe for df\n");
          return(-1);
        }

     while (1)    /* loop until eof on pipe */
      {

       if (fgets(wline1, sizeof(wline1), wfile) == NULL)
        {
         pclose(wfile);
         return(0);
        }

       if ((rc=parse_df_record_sun(wline1, mounted, &size, &free, &used)) == 0)
         {

          for (i=0; i<current_lines; i++)
          {

            if (rc=(strcmp(fs[i], mounted)) == 0)       /* Do we have an exact match? */
              {
               if (used > fsu[i])
                 {
                  first_occur(mounted, used);
                  send_TEC(mounted, size, used, free);
                 }
                break;
              }
```

Figure 151. (Part 4 of 12). Source Listing of fsmon.c

```
            rx=(strspn(fs[i], mounted));
            ptr_fs=fs[i]+rx;
            if (rx >> 0 && (strncmp(ptr_fs, "$", 1) == 0)) /* Do we have a generic match? */
              {
               if (used > fsu[i])
                 {
                  first_occur(mounted, used);
                  send_TEC(mounted, size, used, free);
                 }
                break;
              }
           }
         }
      }
   return(0);
}
```

Figure 152. (Part 5 of 12). Source Listing of fsmon.c

The `validate_usage_sun` routine will issue the df -lk command writing the output to a pipe, read the pipe until EOF, and then call `parse_df_record_sun` for each output line from the df -lk command. There is a test for each found file system if it matches the pattern of the arguments, which the program was called with. There is one test for exact match and another for generic match.

```
/* ************************************************** */
/* Validate usage subroutine for AIX                  */
/* ************************************************** */
int validate_usage_aix()
{

     FILE *tmp1, *tmp2;              /* used for df commands */
     int free, size, used;
     char mounted[PATHMAX];
     int rc=0, rx=0;
     char filesystem[PATHMAX];
     char type[10];
     char cmd[30], wline1[BUFSIZE], wline2[BUFSIZE];

     if ((tmp1=popen("mount", "r")) == NULL)
       {
         fprintf(stderr, "Error when calling the %s command through pipe.\n", "mount");
         exit(-1);
       }

     while (fgets(wline1, sizeof(wline1), tmp1) != NULL)
      {

       if ((rc=sscanf(wline1, "%s %*s %s", filesystem, type) == 2))
        {
         if (strcmp(type, "jfs") == 0 || strcmp(type, "cdrfs") == 0)
          {
           if (strncmp(opsystem, "AIX", 3) == 0)
            {
             if (strcmp(version, "3") == 0)
              {
               sprintf(cmd, "df %s", filesystem);
              }
              else
              sprintf(cmd, "df -k %s", filesystem);
            }

           if ((tmp2=popen(cmd, "r")) == NULL)
            {
             fprintf(stderr,"Error opening pipe for df for AIX\n");
             exit(-1);
            }

           if (fgets(wline2, sizeof(wline2), tmp2) == NULL)
            {
             fprintf(stderr,"Error parsing first header record from pipe for df\n");
             return(-1);
            }

           if (fgets(wline2, sizeof(wline2), tmp2) == NULL)
            {
             fprintf(stderr,"Error parsing df record from pipe\n");
             return(-1);
            }
```

Figure 153. (Part 6 of 12). Source Listing of fsmon.c

```
            pclose(tmp2);

            if ((rc=parse_df_record_aix(wline2, mounted, &size, &free, &used)) == 0)
             {

              for (i=0; i<current_lines; i++)
              {
               if (rc=(strcmp(fs[i], mounted)) == 0)    /* Exact match? */
                 {
                   if (used > fsu[i])
                     {
                      first_occur(mounted, used);
                      send_TEC(mounted, size, used, free);
                     }
                   break;
                 }
               rx=(strspn(fs[i], mounted));
               ptr_fs=fs[i]+rx;

               if (rx >> 0 && (strncmp(ptr_fs, "$", 1) == 0))  /* Do we have a generic match? */
                 {
                   if (used > fsu[i])
                     {
                      first_occur(mounted, used);
                      send_TEC(mounted, size, used, free);
                     }
                   break;
                 }
              }
            }
          }
        }
      }
    pclose(tmp1);
    return(0);
}
```

Figure 154. (Part 7 of 12). Source Listing of fsmon.c

The `validate_usage_aix` routine is similar to the previous for Sun, with one exception: it uses the mount command to find out locally mounted file systems. A df -k will be issued for AIX 4.x and just df for AIX 3.x, as the df command syntax differs between the versions of AIX.

```
/* ************************************************** */
/* Validate usage subroutine for HP-UX                */
/* ************************************************** */
int validate_usage_hp()

{
     FILE *wfile;              /* used for df commands */
     int *pfree, *psize, *pused;
     int free, size, used;
     char mounted[PATHMAX];
     int rc=0, rx=0;
     char cmd[30], wline1[BUFSIZE];
     char *pmounted;
     pmounted=&mounted[0];
     psize=&size;
     pfree=&free;
     pused=&used;

     sprintf(cmd, "df -lk");                          /* df -lk displays only local fs */

     if ((wfile=popen(cmd, "r")) == NULL)
        {
          fprintf(stderr,"Error opening pipe for df\n");
          exit(-1);
        }

     while (1)    /* loop until eof on pipe */
      {

       if (fgets(wline1, sizeof(wline1), wfile) == NULL)
        {
         pclose(wfile);
         return (0);
        }

       rc=0;
       rx=sscanf(wline1, "%s %*s %*s %*s %d %*s %*s %*s",
                       pmounted, &*psize);
       rc=rc+rx;
       if (fgets(wline1, sizeof(wline1), wfile) == NULL)
         {
           fprintf(stderr,"Error parsing HP-UX df command from pipe\n");
           return(-1);
         }

       rx=sscanf(wline1, "%d %*s %*s %*s",
                       &*pfree);
```

Figure 155. (Part 8 of 12). Source Listing of fsmon.c

```
       rc=rc+rx;

       if (fgets(wline1, sizeof(wline1), wfile) == NULL)      /* skip used allocated Kb   */
         {
           fprintf(stderr,"Error parsing HP-UX df command from pipe\n");
           return(-1);
         }

       if (fgets(wline1, sizeof(wline1), wfile) == NULL)
         {
           fprintf(stderr,"Error parsing HP-UX df command from pipe\n");
           return(-1);
         }

       rx=sscanf(wline1, "%d %*s %*s %*s",
                       &*pused);
       rc=rc+rx;

       if (rc == 4)
         {
          for (i=0; i<current_lines; i++)
          {

            if (rc=(strcmp(fs[i], mounted)) == 0)       /* Exact match? */
              {
               if (used > fsu[i])
                 {
                  first_occur(mounted, used);
                  send_TEC(mounted, size, used, free);
                 }
                break;
               }
              rx=(strspn(fs[i], mounted));
              ptr_fs=fs[i]+rx;
              if (rx >> 0 && (strncmp(ptr_fs, "$", 1) == 0)) /* Do we have a generic match? */
                {
                if (used > fsu[i])
                  {
                   first_occur(mounted, used);
                   send_TEC(mounted, size, used, free);
                  }
                  break;
                }
            }
         }
      }
   return(0);
}
```

Figure 156. (Part 9 of 12). Source Listing of fsmon.c

The `validate_usage_hp` routine differs from the other two in that a df -lk command on the HP-UX platform results in multiple lines per entry as shown in Figure 145 on page 155. Considering this, the `validate_usage_hp` extracts data from the df -lk in another way.

```
/* ************************************************** */
/* Parse df command output for SunOS                  */
/* ************************************************** */
int parse_df_record_sun(line, mounted, size, free, used)
char *mounted;
int *size, *free, *used;
{

  int rc;

  rc=sscanf(line, "%*s %d %*d %d %d%% %s",
                       &*size, &*free, &*used, mounted);

  switch (rc)
  {
    case 4:      /* got all values */
      return(0);
    case EOF:
      perror(prog);
      return(-1);
    default:  /* header line or record with undefined values */
      return(1);
      break;
  }

}

/* ************************************************** */
/* Parse df command output for AIX                    */
/* ************************************************** */
int parse_df_record_aix(line, mounted, size, free, used)
char *mounted;
int *size, *free, *used;
{

  int rc;

  rc=sscanf(line, "%*s %d %d %d%% %*s %*s %s",
                      &*size, &*free, &*used, mounted);

  switch (rc)
  {
    case 4:      /* got all values */
      return(0);
    case EOF:
      perror(prog);
      return(-1);
    default:  /* header line or record with undefined values */
      return(1);
      break;
  }
}
```

Figure 157. (Part 10 of 12). Source Listing of fsmon.c

```
/* ************************************************** */
/* get hostdata   (hostname, opsystem)            */
/* ************************************************** */

int get_hostdata(void)
{
   int rc;
   char hcmd[8];
   char tline1[80];
   FILE *tmp1;
   sprintf(hcmd, "uname -a");
   if ((tmp1=popen(hcmd, "r")) == NULL)
       {
        fprintf(stderr,"Error opening pipe for get hostdata\n");
        exit(-1);
       }
   if (fgets(tline1, sizeof(tline1), tmp1) == NULL)
      {
       fprintf(stderr,"Error parsing line for uname -a from pipe\n");
       exit(-1);
      }
   pclose(tmp1);
   rc=sscanf(tline1, "%s %s %s %s %*s",
                    opsystem, hostname, rel, version);

   switch (rc) {
    case 4:      /* got all values */
      return(0);
    case EOF:
      perror(prog);
      return(-1);
    default:  /* header line or record with undefined values */
      return(1);
      break;
  }
}
```

Figure 158. (Part 11 of 12). Source Listing of fsmon.c

The `get_hostdata` routine issues the uname -a command and extracts platform-specific information. Output from the uname command is shown in Figure 160 on page 172.

```
/* ************************************************** */
/* set Tivoli environment                             */
/* ************************************************** */
void set_Tivoli_env()
{
   char tcmd[25];
   FILE *tmp2;
   sprintf(tcmd, ". /etc/Tivoli/setup_env.sh");
   if ((tmp2=popen(tcmd, "r")) == NULL)
       {
       fprintf(stderr, "Error opening pipe for set Tivoli environment\n");
       }
   pclose(tmp2);
}
/* ************************************************** */
/* send TEC event subroutine                          */
/* ************************************************** */
void send_TEC(mounted, size, used, free)
char *mounted;
int *size, *used, *free;
{
   FILE *tecfile;
   sprintf(eventmsg, "postemsg -S %s -r %s -m %s hostname=%s filesystem=%s fssize=%d
    fsusage=%d fsfree=%d fsthreshold=%d FSMON_mon FSMON", TECSERVER, sevflag, mounted,
    hostname, mounted, size, used, free, fsu[i]);
   if ((tecfile=popen(eventmsg, "r")) == NULL)
       {
        fprintf(stderr, "Error opening pipe for TEC event");
       }
   pclose(tecfile);
}
/* ************************************************** */
/* Prepare message with node, filesystem, and usage.  */
/* When program exits, this message will be printed.  */
/* ************************************************** */
void first_occur(mounted, used)
char *mounted;
int *used;
{
if (strlen(alertmsg) > 940)
  {
   fprintf(stderr, "Alert message longer than 940 bytes");
   exit(-1);
  }
char alertwrk[80];
sprintf(alertwrk,"Node=s%_fs=%s_usage=%d ", hostname, mounted, used);
strncpy(alertmsg+strlen(alertmsg), alertwrk, strlen(alertwrk));
first_occ = 1;
}
```

Figure 159. (Part 12 of 12). Source Listing of fsmon.c

```
AIX rs600019 2 4 003C30244C00

SunOS sun 5.5.1 Generic sun4m sparc SUNW,SPARCstation-20

HP-UX utb11 B.10.01 A 9000/856 1135310352 two-user license
```

Figure 160. Output from uname -a Command

3.13.3 New TEC Definitions

We decide to add a new TEC event class. This is done because we want to have a new source, so we can see that events are coming from our fsmon program. We also want to add some new fields, which do not exist in the default TEC class definitions. Furthermore, we want to delete duplicate events if they occur within 3660 seconds from the original. To accomplish all this we need two TEC definitions:

- A class definition

  The baroc file defines the new event class.

- A rule

  The rule tells the TEC server how to act when new events occur.

```
TEC_CLASS:
    FSMON_mon ISA EVENT
        DEFINES {
                 source: default = "FSMON";
                 hostname: STRING, dup_detect = yes;
                 filesystem: STRING, dup_detect = yes, default = "/";
                 fssize: INTEGER;
                 fsfree: INTEGER;
                 fsthreshold: INTEGER;
                 fsusage: INTEGER;
                 severity: default = WARNING;
                };
END
```

Figure 161. Listing of fsmon.baroc

The fsmon.baroc file shown in Figure 161 is used to define a new source FSMON, and some new fields, called slots. The new slots are:

- filesystem
- fssize
- fsfree
- fsthreshold
- fsusage

When issuing the postemsg command, we reference these new slots.

```
rule: dup_FSMON_mon:
(
      event: _event of_class
            'FSMON_mon',
      action: dup_and_drop_event:
          (
           first_duplicate(_event, event: _dup_FSMON_ev
                                where [status: outside ['CLOSED'],
                                       filesystem: equals _filesystem,
                                       hostname: equals _hostname
                                      ],
                            _event - 3660 - 3660
                           ),
           add_to_repeat_count(_dup_FSMON_ev, 1),
           drop_received_event
          )
).
```

Figure 162. Listing of fsmon.rls

The fsmon.rls file shown in Figure 162 is a new rule we implement to take care of duplicate events. The rule will check for events that are not closed and where the file system and hostname already exists in an older event. If a match is found within 3660 seconds from the original event, the repeat_count will be updated and the new event is dropped.

3.13.4 Adding a New Rule Base

We need to add our TEC definitions to the TEC Event Server. This is done in the following steps:

1. Create a new rule base DM36:

   ```
   wcrtrb -d $BINDIR/TME/TEC DM36
   ```

2. Copy all rules and class definitions from the Default rule base or the currently active rule base:

   ```
   wcprb Default DM36
   ```

3. Compile the DM36 rule base and load it:

   ```
   wcomprules DM36
   wloadrb DM36
   ```

   You should then have two rule bases as shown in Figure 163 on page 174. DM36 is the active one, as indicated by the arrow.

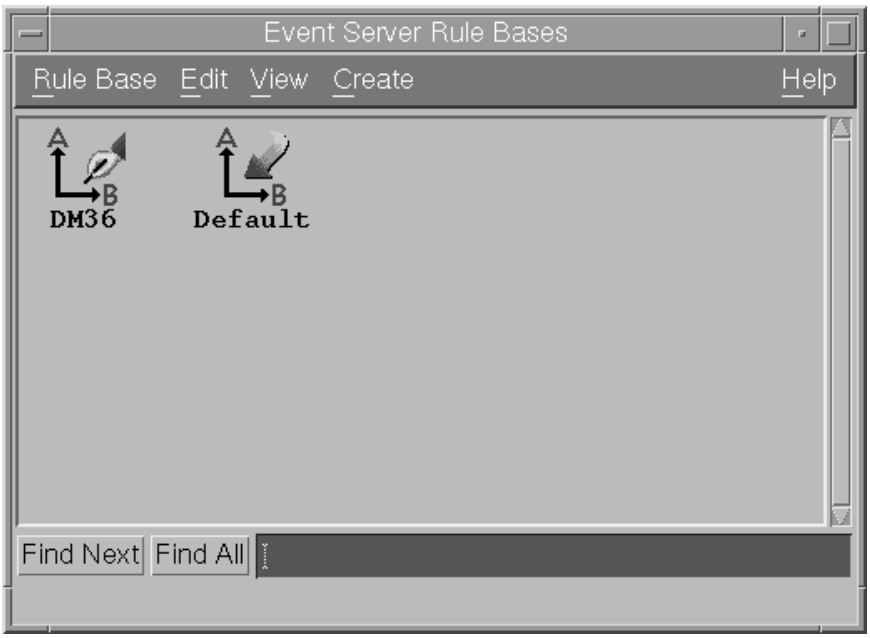


Figure 163. Rule Bases

4. Click on the right button on DM36 and select **Import...** from the pull-down menu.

   You will see the window as shown in Figure 164 on page 175. Select **Import Class Definitions**, **Insert After** and click on **File...**.

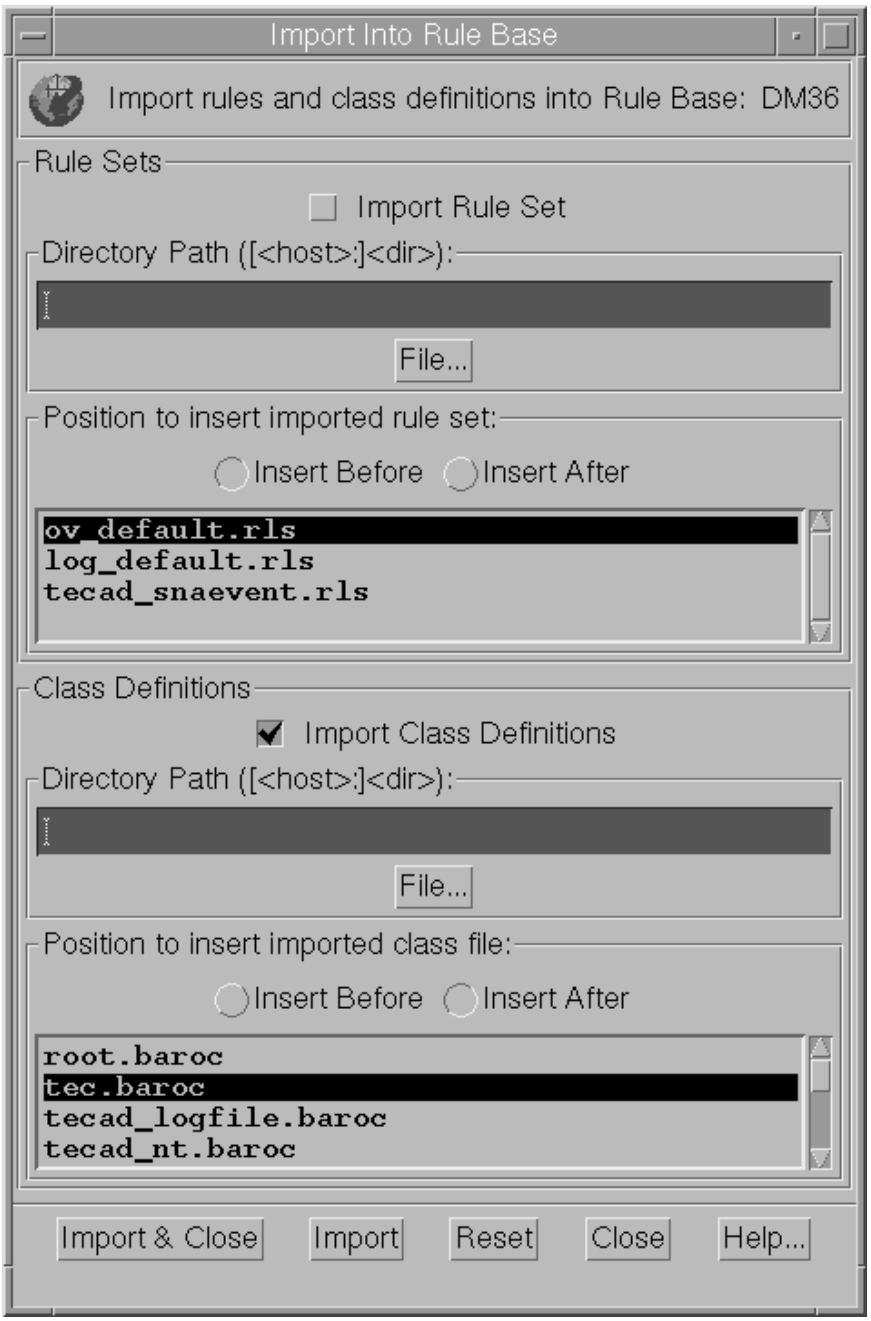


Figure 164. Import Into Rule Base

5. Find the fsmon.baroc file and select it as shown in Figure 165 on page 176.

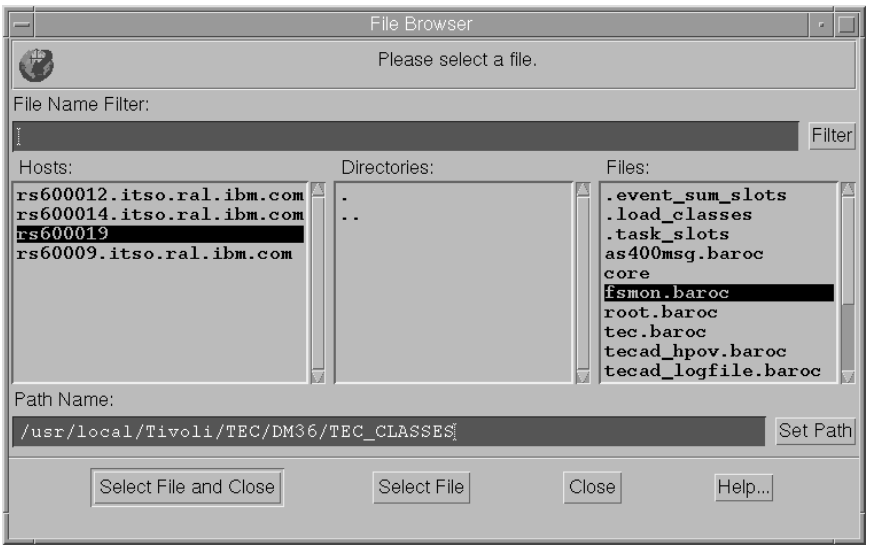


Figure 165. Import Rule Base - File Browser

Click on **Select File and Close**.

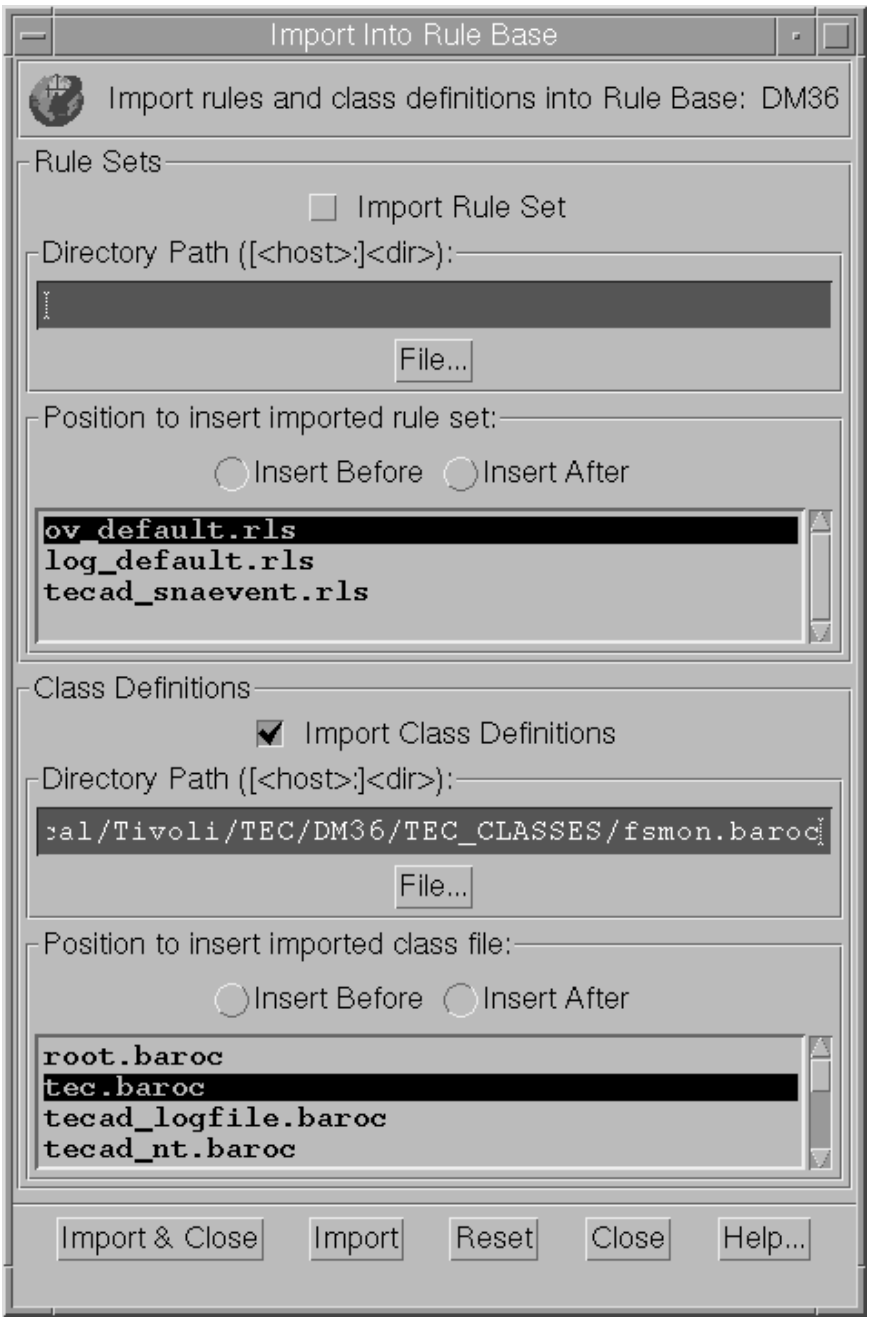


Figure 166. Import Into Rule Base - File Specified

6. Now select **Import**.
7. Do the same import actions for the rule set, and import fsmon.rls.
8. Now that you have imported the fsmon definitions you have to compile your rule base. The output from the compilation is shown in Figure 167 on page 178.
9. Load the rule base.
10. Restart the TEC Event Server.

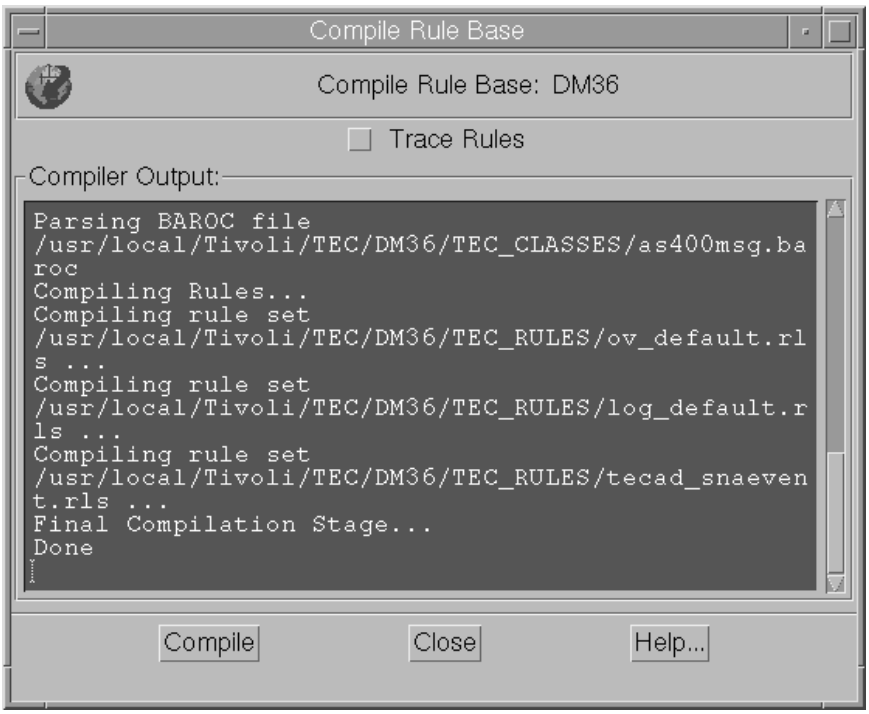


Figure 167. Compile Rule Base

Now the final step is to load the DM36 rule base into the event server.

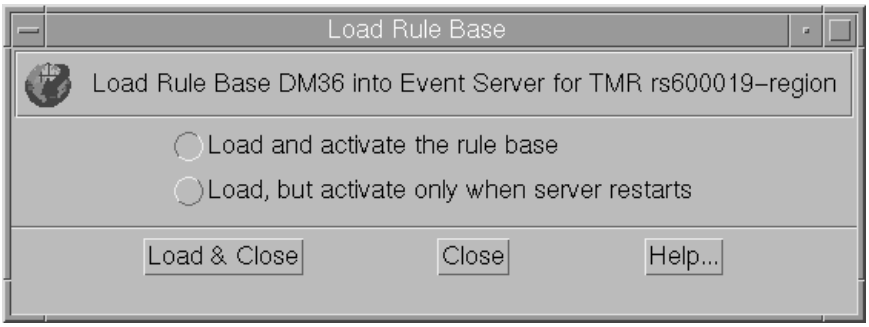


Figure 168. Load Rule Base

We now have an active rule base called DM36 that will recognize our new fsmon events.

Before we are finished with preparing the TEC for our program, we need to do a few more things:

- Add a new source to the TEC server.

  This is done by clicking with the right button on the **EventServer** icon and selecting **Sources**. The window shown in Figure 169 on page 179 appears once you have added the name and label FSMON for the new source:

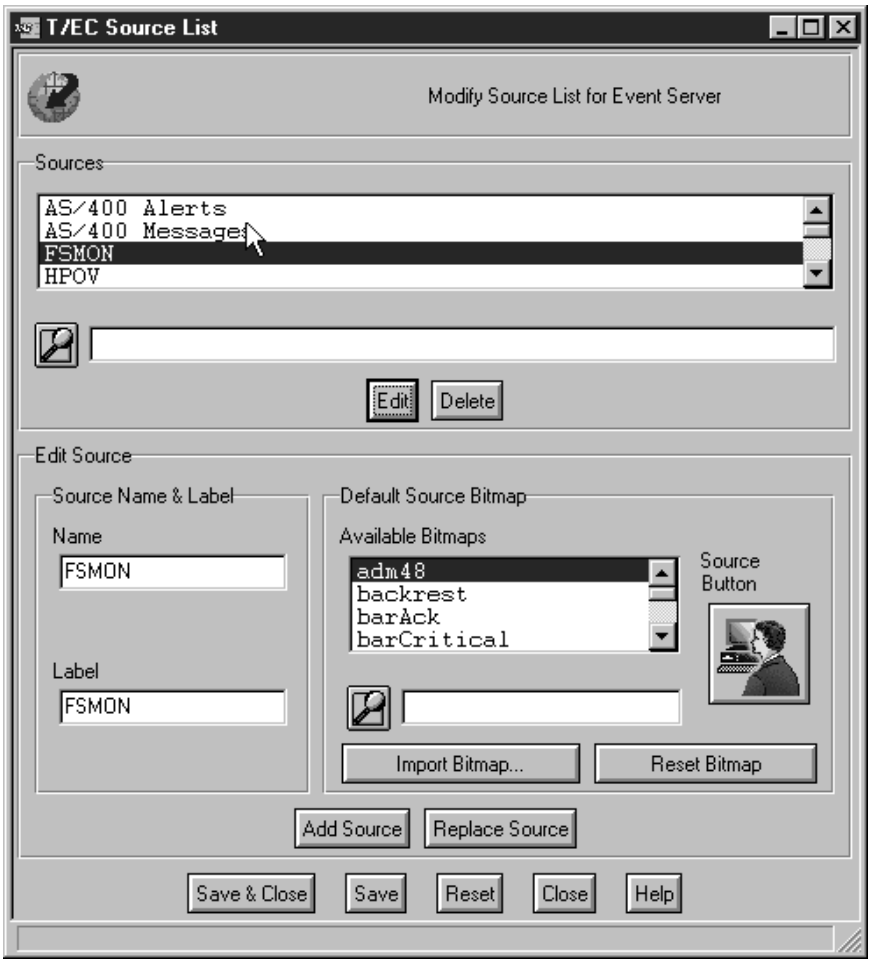


Figure 169. T/EC Source List Window

- Add a new event group to the TEC server.

  This is done by clicking with the right button on the **EventServer** icon and selecting **Event Groups**. Then you have to add a filter in this event group to set the source to FSMON, as shown in Figure 170 on page 180:

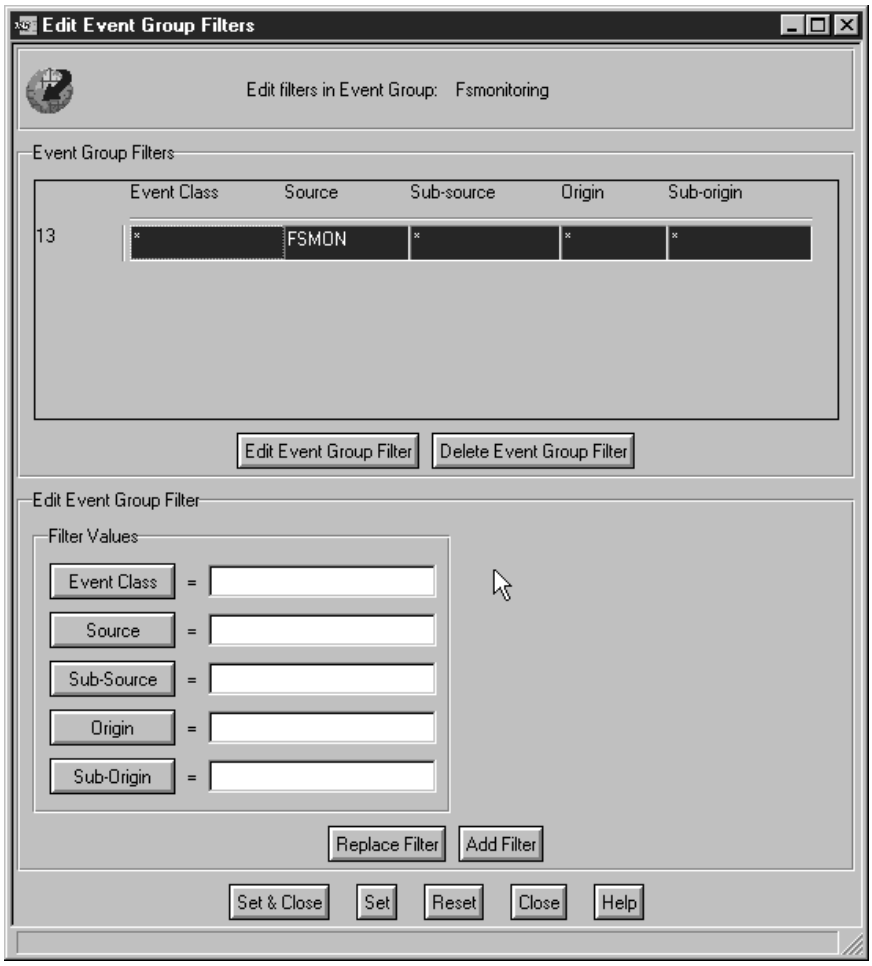


Figure 170. Edit Event Group Filters Window

- Assign the new event group to an administrator.

  This is done by clicking with the right button on the event console icon and selecting **Assign Event Groups**.

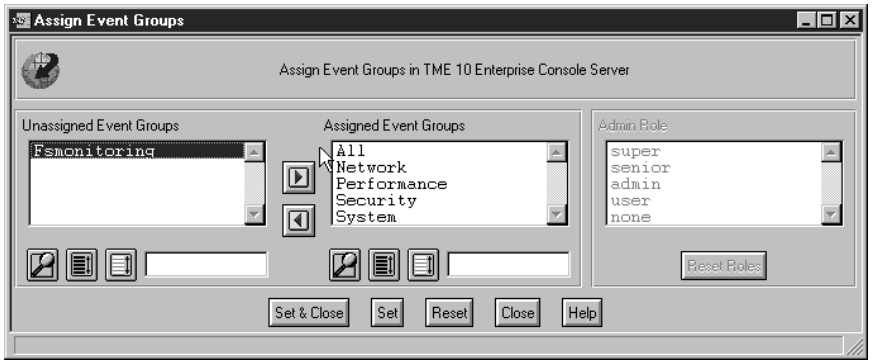


Figure 171. Assign Event Groups Window

3.13.5 The FSMON Program in Action

We will now test our new monitoring collection fsmon. To do this we first create an indicator collection FSMon and a dataless profile manager FSMon_test_PM as shown in Figure 172.

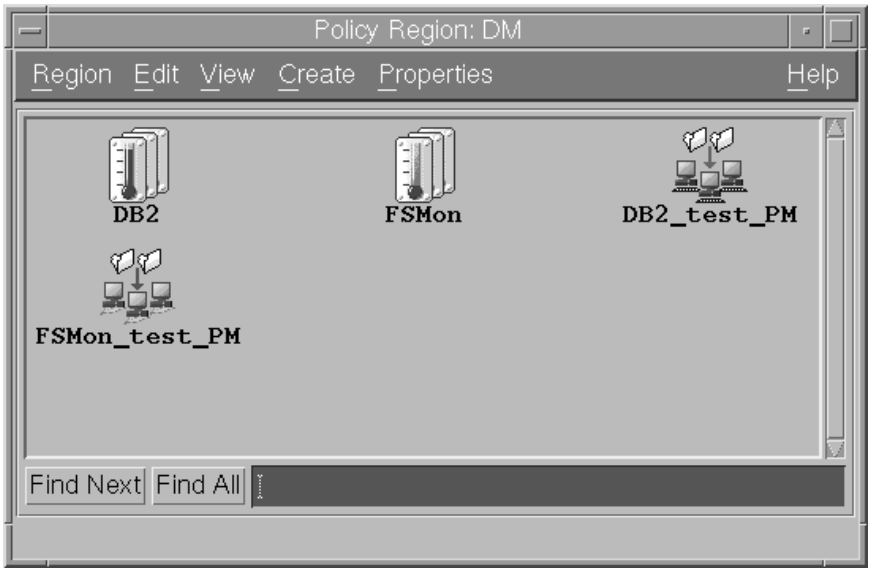


Figure 172. Policy Region DM

Now we open the profile manager by double-clicking on the **FSMon_test_PM** icon and add the LCF endpoints as subscribers, rs600019, hp, and sun. We add a profile called **PH2**. Refer to Figure 173 on page 182 for what it will look like when done.

Figure 173. Profile Manager - FSMon_test_PM

Next, we add a monitor to the profile PH2, so we open the profile and select **Add Monitor...**.

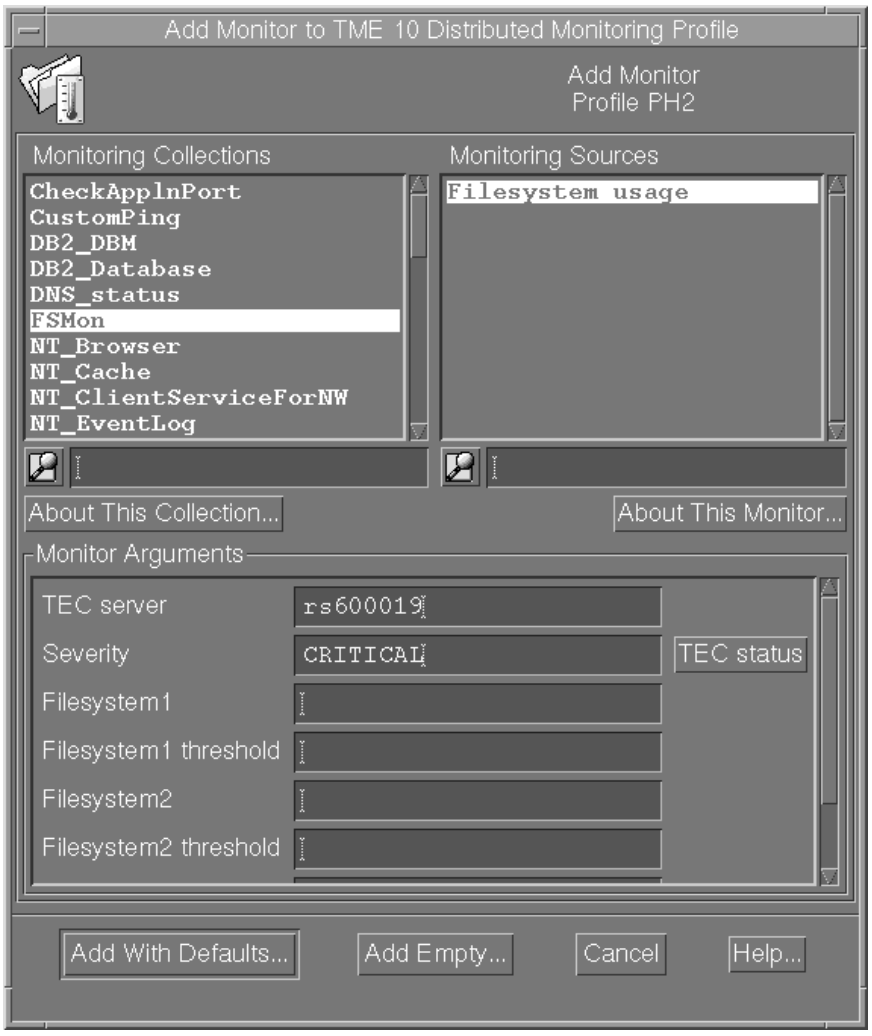


Figure 174. FSMon - Add Monitor

In the Add Monitor to Tivoli Distributed Monitoring Profile window in Figure 174 we select the **FSMon** Monitoring Collection and the **Filesystem usage** Monitoring Source. The default monitor arguments are filled in for us. We want to change the severity from CRITICAL and click on **TEC status** and select **WARNING** in the window shown in Figure 175 on page 184.

The severity is the TEC event severity when the event is presented on the TEC console. As you can change this for different file systems you can, for instance, have two monitors on one node, where one monitor results in warnings and the other in critical events.

Figure 175. FSMon - Severity

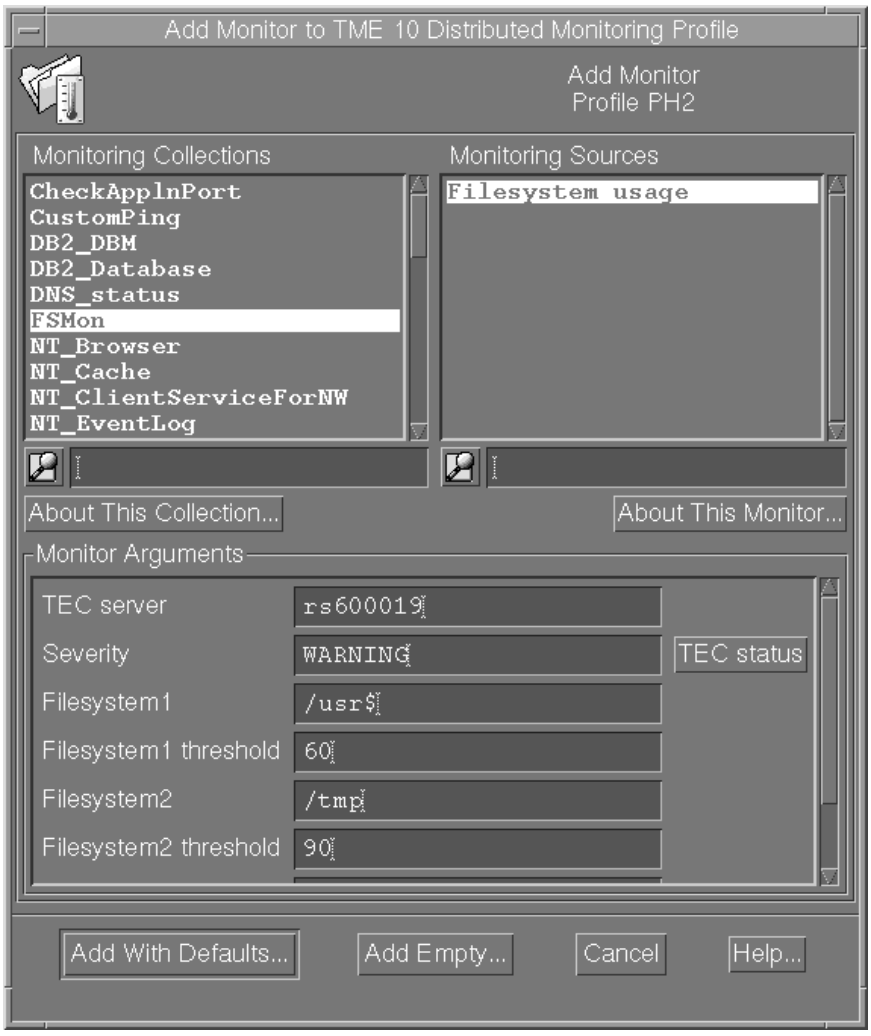


Figure 176. FSMon - Arguments

In the window shown in Figure 176 we add /usr$ with a threshold of 60%, and /tmp with a threshold of 90%. Then we select **Add Empty...**.

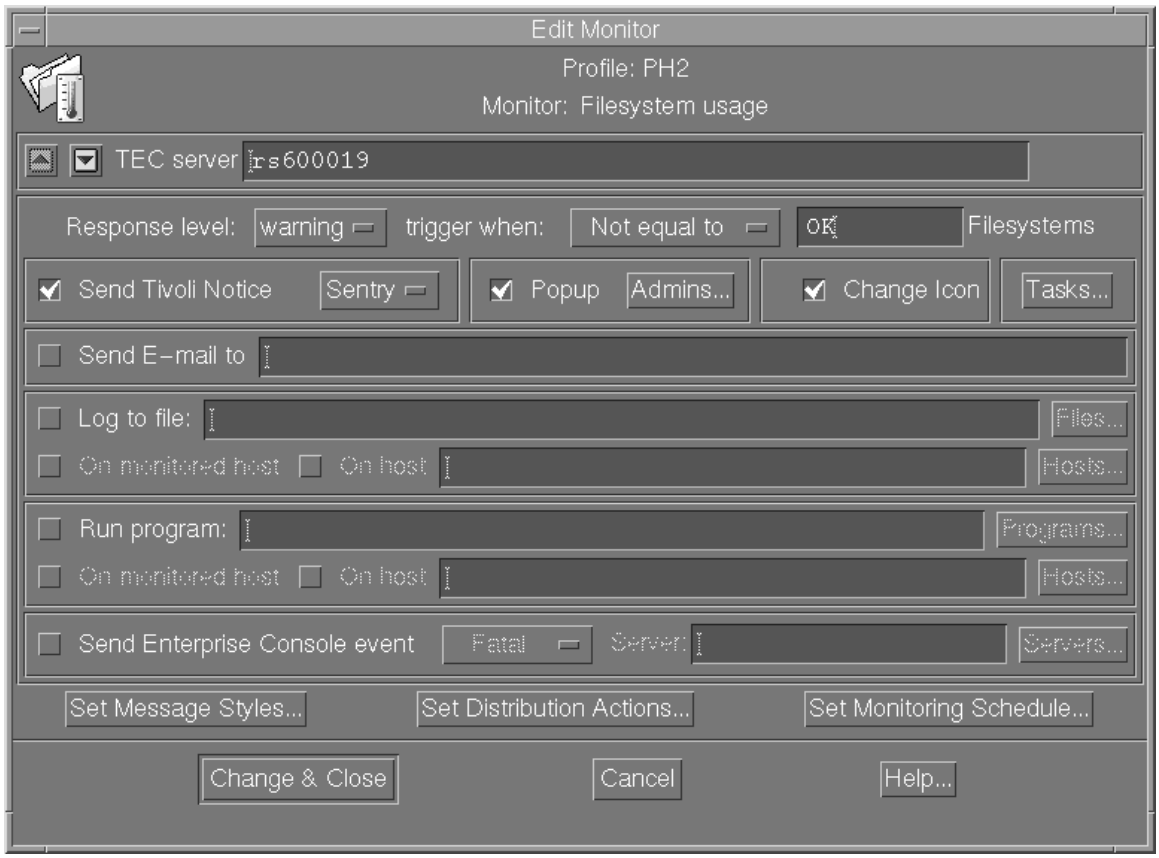


Figure 177. FSMon - Edit Monitor

The window in Figure 177 is displayed and we select Response level **warning**, **Not equal to** and 0K in the Not equal to box. We also select for presentation **Send Tivoli Notice**, **Popup**, and **Change Icon**. Then we select **Change & Close**.

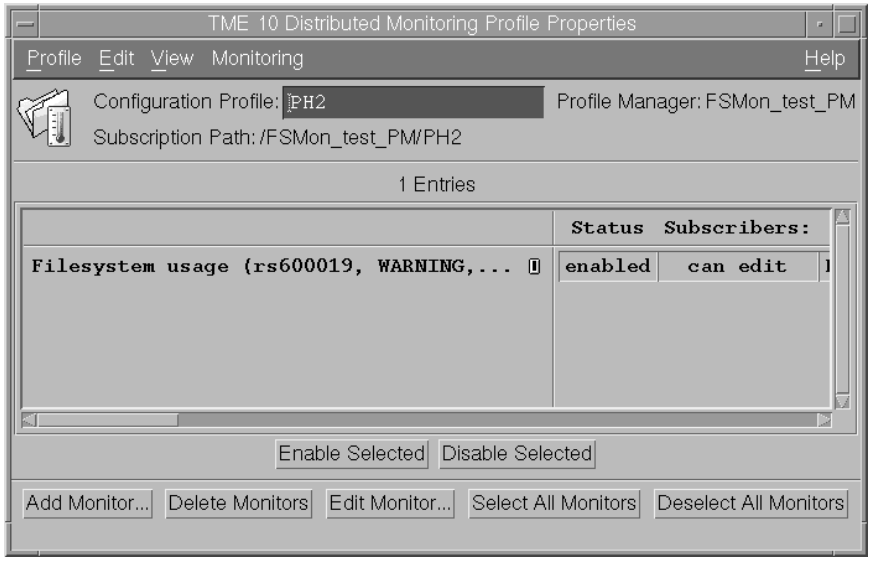


Figure 178. FSMon - Monitor Properties

The window in Figure 178 is displayed. We select **Profile** from the menu bar and then **Save** from the pull-down menu. Then we select **Profile** from the menu bar again and then **Exit** from the pull-down menu.

Now we distribute the profile to our LCF endpoint node rs600019. When the monitor begins you will see the pop-up as in Figure 179.

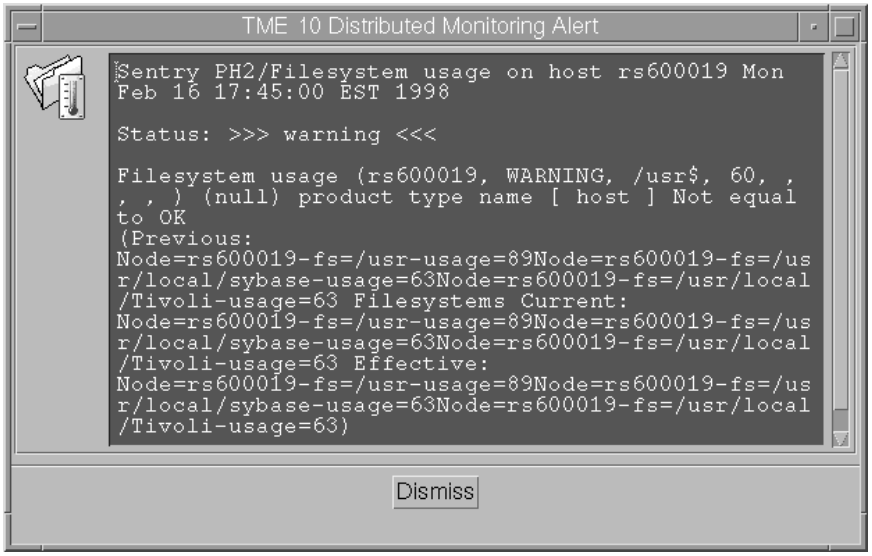


Figure 179. FSMon - Pop-up

As you can see there are three file systems that exceed the threshold of 60%:

- /usr 89%
- /usr/local/Tivoli 63%
- /usr/local/sybase 63%

The same information can be seen in TEC as shown in Figure 180 where we can see a **WARNING** situation exists in the **FSMonitor** event group.

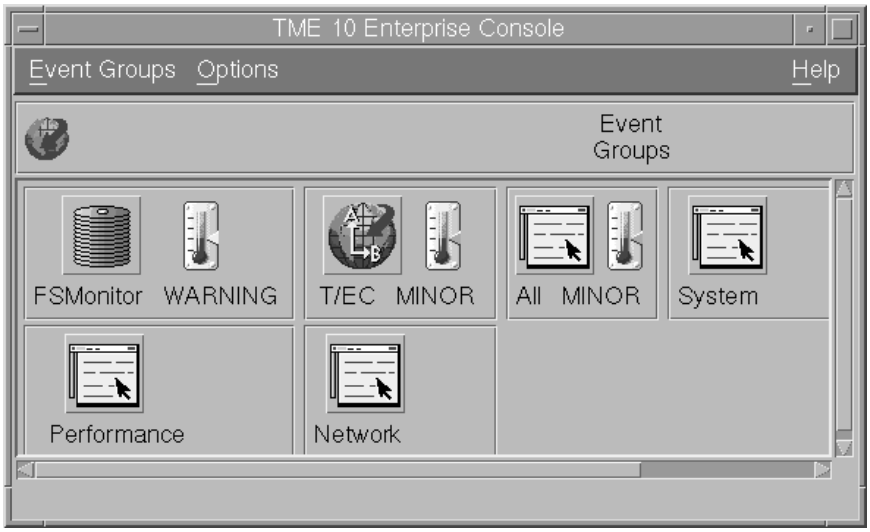


Figure 180. FSMon - TEC Console

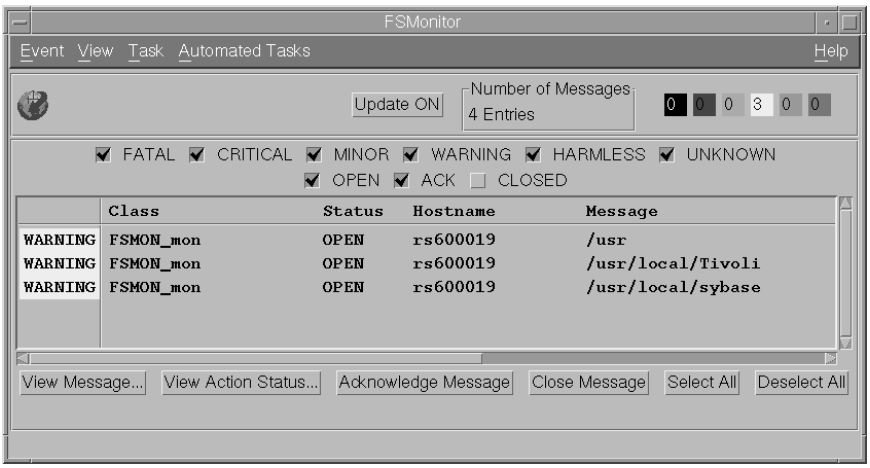


Figure 181. FSMon - Events

In Figure 181 we have opened the FSMonitor event group and can see the three events posted by the fsmonaix program. By selecting an event and clicking on **View Message...** we are presented with the window in Figure 182 on page 189, which is a very detailed display of the event.

Here you can also see that we have a **repeat_count=5**, which means our rule is working and adding 1 to the repeat count for duplicates.

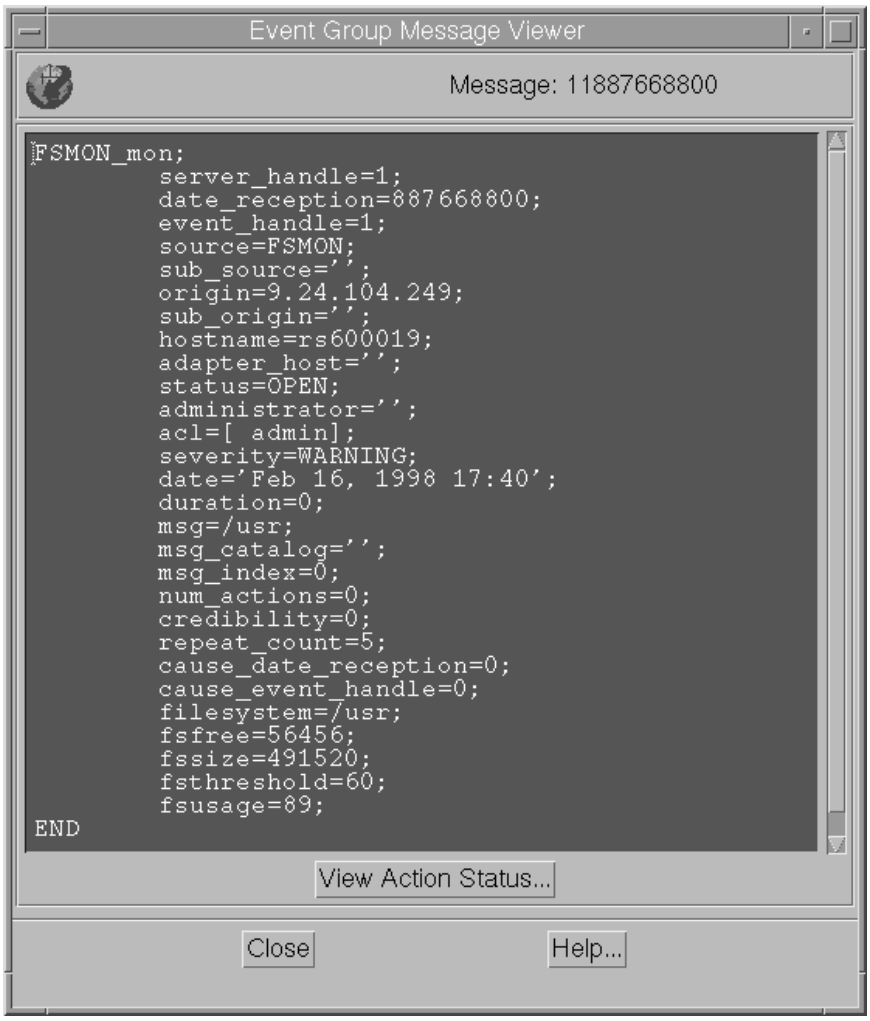


Figure 182. FSMon - Event Details

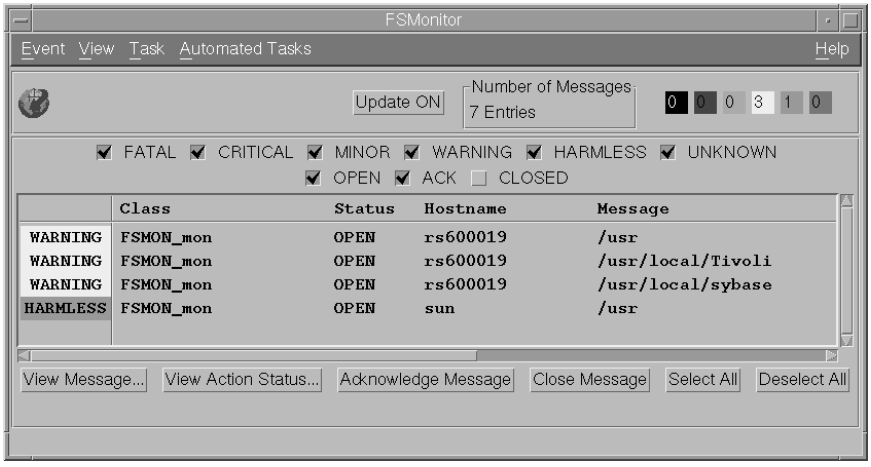


Figure 183. FSMon - Events from Multiple Sources

The last display in Figure 183 shows two different severities from sun and rs600019.

3.13.6 File Systems Monitoring Summary

We now have a monitor in a monitoring collection for file system monitoring. This monitor should provide a good base for further customization of Tivoli Distributed Monitoring with regard to file systems.

Chapter 4. Creating a Tivoli Installable Image for our Custom Monitors

In this chapter, we describe the steps needed to produce an installable package for our custom monitors. This package will be named "TME 10 Distributed Monitoring ITSO Collection 1".

It will be installed from the Tivoli desktop as shown in the following figure.

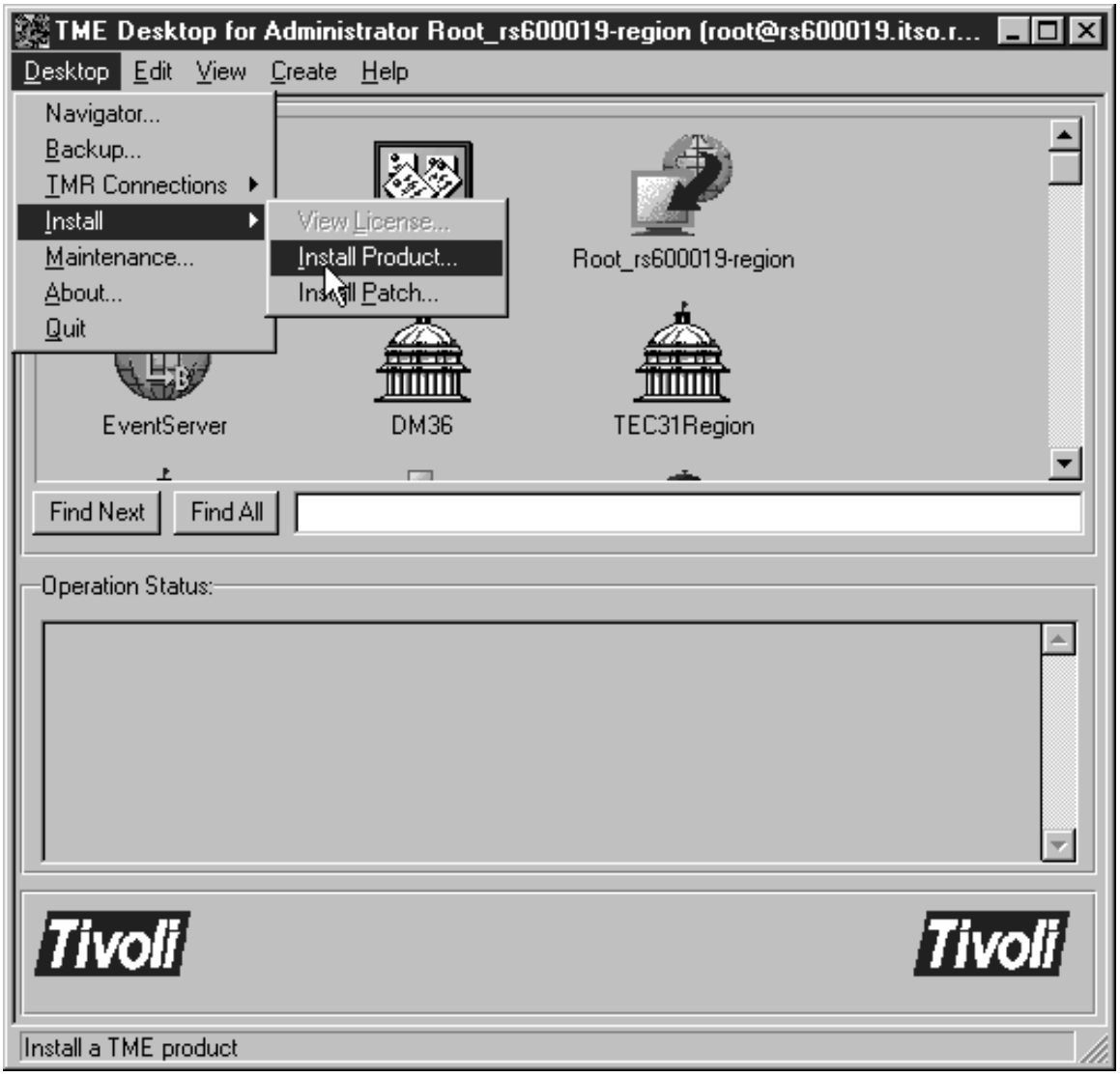


Figure 184. Install Product Pull-Down Menu

The installation procedure will automatically copy the collection files into the Tivoli directory and integrate them in the Tivoli Distributed Monitoring environment, the same way we installed the UNIX, NT and Universal Monitoring Collections.

4.1 Overview and Objective

The Tivoli Framework provides a simple mechanism to push TME 10 components and management applications to the systems being managed. We will use these installation routines to install our monitoring collection on our TMR server. This mechanism is independent of the platform; whether NT or UNIX, the procedure is identical. This is one of the advantages of the Tivoli Framework features.

4.2 Prerequisites

It is necessary to install the product Tivoli ADE (Application Development Environment) prior to starting the creation of your installation image. The commands wcrtindex and wcrtimage come with ADE. These commands are used to automatically create some files in a format specific to the Tivoli Framework.

4.3 Creating the Installation Image

In this section we describe step by step how we create our image. There are five steps in creating this package:

1. Preparing the collection files.
2. Preparing the before and after scripts.
3. Defining the layout of the application image.
4. Creating the installation image from the image definition.
5. Checking the image contents.

Basically, the installation process is performed in two passes:

- The first pass consists of pushing the collection files to the Tivoli destination directory. This operation is named "Binaries" (BIN).
- The second pass consists of running an after script and is named "Database Server" (ALIDB).

The files we have to prepare will refer to these two passes.

4.3.1 Preparing the Collection Files

First, we create a directory to put the installable images in. In our example we use:

```
mkdir /:/project/distmon/tme_install
```

Note

The `/:/` means we use a DFS (Distributed File System). Our working directories will always start with a `/:/`.

Then we copy the `.col` files we want to install to a specific directory. This directory will be used as a source to populate the image. We check the contents of this source directory with the command:

```
ls -l /:/project/distmon/code/SentryMonitors:

-rw-rw-r--   1 stefanu  assignee   21289 Mar 03 12:44 cpu.col
-rw-rw-rw-   1 stefanu  assignee     266 Mar 03 12:44 fsmon.baroc
-rw-rw-rw-   1 stefanu  assignee     972 Mar 03 12:44 fsmon.rls
-rw-rw-r--   1 stefanu  assignee   12535 Mar 03 12:44 fsmond.col
-rw-rw-r--   1 stefanu  assignee   11945 Mar 03 12:44 mon8235.col
-rw-rw-r--   1 stefanu  assignee     117 Mar 03 12:44 monvm.baroc
-rw-rw-r--   1 stefanu  assignee   13007 Mar 03 12:44 monvm.col
-rw-rw-r--   1 stefanu  assignee   14334 Mar 03 12:44 snapd.col
-rw-rw-r--   1 stefanu  assignee   14761 Mar 03 12:44 snapm.col
-rw-rw-r--   1 stefanu  assignee    8461 Mar 03 12:44 tmon.col
```

4.3.2 Preparing the After Scripts

An after script will be executed after pushing the collection files to the target directory. Basically, this script will perform three operations:

1. Install the new collections on the TMR server (mcsl -R -i).
2. Restart the Sentry engine to load the new collections (mcsl -R).
3. Report any error if needed in temporary files under /tmp.

As a starting point, we use the after scripts provided by the UNIX Monitoring Collection and we have adapted them to our needs. We have decided to develop the installation script for UNIX only, but it is possible to create different packets for each interpreter type. We have created a subdirectory in the install directory. We check its contents with the ls command:

```
ls -l /:/project/distmon/install/ITSOMON_scripts:

-rw-r--r--   1 stefanu  assignee    3177 Mar 03 12:52 ALIDB_after_nt.sh
-rw-r--r--   1 stefanu  assignee    3075 Mar 03 12:52 ALIDB_after_unix.sh
```

See Figure 185 on page 194 for the shell script we use as an after script for the Server Database (ALIDB) part of our installation. You can see that this script just creates temporary files for output and performs an mcsl - R -i for each collection file. Then it restarts the Sentry engine to load the new collections.

```
#!/bin/sh
#
# This is a patch after script,  generated by the patchgen script.
# You should not edit this file.
#
_TMP="$TMP"
if [ "x$_TMP" != x ]; then
    # Unscrew NuTcracker screwyness.
    _TMP= echo "$_TMP" ] sed 's!/\([a-zA-Z]\)=/!\1:/!g'
    if [ ! -d "$_TMP" ]; then
unset _TMP
    fi
fi
if [ "x$_TMP" = x ]; then
    # This should be a no-op on Unix, but on NT the chances are good that
    # there really will be a $DBDIR/tmp.
    if [ -d tmp ]; then
_TMP= pwd /tmp
    else
_TMP=/tmp
if [ ! -d /tmp ]; then
    mkdir /tmp
fi
    fi
fi
export _TMP

if [ -f /tmp/TransferEnv ] ; then
    set -a
    . /tmp/TransferEnv
fi

PatchId=ITSOMON_ALIDB_$1
PATH=${PATH}:/usr/ucb

if [ -d "$ThisDir" ]; then
    cd $ThisDir
fi

sh >$_TMP/${PatchId}.output 2>$_TMP/${PatchId}.error <<\MAIN_EOF
set -xe

#
# Get the NameRegistry, InterfaceRepository and Library object IDs
# If the appropriate env variables are already set, then we
# don't need to get them again.  Additionally, if we need to get
# the InterfaceRepository oid, go ahead and put it into replace mode.
#
NameRegistry=${NameRegistry:- wlookup NameRegistry }
export NameRegistry
TNR=$NameRegistry
export TNR
```

Figure 185. Listing of UNIX After Script (Part 1 of 3)

```
Library=${Library:- wlookup Library }
export Library
CLO=$Library
export CLO
if [ -z "$InterfaceRepository" ] ; then
InterfaceRepository= wlookup InterfaceRepository
        export InterfaceRepository
idlattr -s -t $InterfaceRepository replace_mode boolean TRUE
fi

################################################
# /tivoli/builds/1/TMP_3.2/src/lcf-world/lcf/tests/images/patchgen.pl generated after script
# PatchId: ITSOMonitors36
#
# Executing after script:  TME/SENTRY/tivolicol.install
sh -xe <<\ALIDB_EOF

#!/bin/sh
############################################################
#
# Installation script for the Tivoli monitoring collection
#
# Mike McNally
#
# $Id: tivolicol.install,v 1.7 1996/07/11 15:00:11 ccorley Exp $
#

#exec >> /tmp/sentry2_0.mon.sinst 2>&1
echo "================================================"
date
set -x
#
# Figure out what the deal is
#
#eval  odadmin ] awk '
# /[Interpreter/ {printf "arch=%s\n", $NF}
# /[Install directory/ {printf "bindir=%s\n", $NF}
#'
OOID= objcall 0.0.0 get_oserv
bindir= objcall $OOID query install_dir ] tr '\\\\' '/'
#
# Drop down into the collections directory
#
cd ${bindir}/generic/SentryMonitors
#
# Install the collection
#
```

Figure 186. Listing of UNIX After Script (Part 2 of 3)

```
set -e
${bindir}/${INTERP}/bin/mcsl -R -i cpu.col
${bindir}/${INTERP}/bin/mcsl -R -i fsmond.col
set +e
#
# Create default monitors for this collection
#
#sh -x defaults.sh

exit $?

exit 0

ALIDB_EOF

MAIN_EOF

SSSTAT=$?
# any output needs to go to stderr

if [ $SSSTAT -ne 0 ]; then
    echo "" >&2
    echo "Failure: the last few lines of the error log ($_TMP/$PatchId.error):" >&2
    tail $_TMP/$PatchId.error >&2
    if [ ! -z "$FileIfInstalled" ] ; then
    rm -f $FileIfInstalled
    fi
else
    echo "Product install completed successfully." >&2
    msg="";
    if [ -n "$msg" ]; then echo " $msg" >&2; fi

    rm -f $_TMP/$PatchId.error $_TMP/$PatchId.output
fi
exit $SSSTAT
```

Figure 187. Listing of UNIX After Script (Part 3 of 3)

You can see the mcsl commands in Part 3 of the listing of our after script.

4.4 Preparing the Layout of the Installation Image

This step consists of creating an index file (`.IND`), a cross reference file (`.XREF`) and configuration (`.CFG`) files. The `CFG` files will be located in the CFG subdirectory of the installation image. This set of files is the layout of our installation image. The command wcrtindex (this command comes with Tivoli ADE) is useed to create this layout. It is an interactive command. It prompts you to define some tags, variables and data that will be used by the installation routines. You can run the command as follows:

```
wcrtindex /:/project/distmon/install ITSOMON 2000
```

You will be prompted to enter some information. This information will be saved in a file wcrtindesx.log under /tmp. You can use this log file later as a standard input for the wcrtindex command.

```
Enter a product tag (i.e. TMF, SENTRY, COURIER, etc.)
This will be the starting tag for all lines in
/:/project/distmon/install/ITSOMON.IND (just hit return to use
ITSOMON)
```

Press the Return key to use the default.

```
Enter the full name of the product referenced by the
product tag "ITSOMON":
```

Enter the name of the product you want to create, in our case:

```
TME 10 Distributed monitoring ITSO Collection1

Enter the product licensing tag (TIV_AEF, TIV_ADE...)

Unless you enter a string here, ITSOMON will be used
```

Press the return key to accept the default value.

```
Enter an id tag (e.g. BIN, LIB, DB...).
This will be the tag that your install packets will
reference (Return to continue) :

BIN

Enter the full name of this id tag :

Binaries
Is Binaries used for client (c), server (s), or both (b) :
s

Enter the value of the sub directory path (the path that
gets appended to the path the user will enter) for Binaries
(e.g. @Arch@, @HostName@.db)(Press return to skip) :

generic

Enter the default installation path or a comma separated list of choices
 for Binaries (Press Return to skip):

/usr/local/Tivoli/bin

Enter optional environment variables needed
(e.g. @Env@)(Press return to skip) :

Enter id to chain from. (e.g. BIN, LIB, ...)
(Press return to skip) :

Creating the install packet/script entries for Binaries.

Select install packet (f) or script (s) (Press Return to continue) : f

First we need to know whether to create a install packet for
each each interptype or just one install packet.
```

```
Enter a '+' to indicate a per interptype install packet
or enter the desired contents for the arch field :
generic

Enter the path to the files to put in the install packet :
/:/project/distmon/code/SentryMonitors

Enter the path to a before script (Type Enter to skip) :

Enter the path to an after script (Type Enter to skip) :

Enter the name of a file (a perl regex) to look for to decide if this por
is installed.  You can also add to the disk space requirements for insta
this package by adding a ; followed by a number after the file to look f
For example: fileB;1024 would specify to look for a file named fileB and
add a check for an additional 1024K of disk space (Type Enter to skip) :

Creating the install packet/script entries for Binaries.

Select install packet (f) or script (s) (Press Return to continue) :

Enter an id tag (e.g. BIN, LIB, DB...).
This will be the tag that your install packets will
reference (Return to continue) :
ALIDB

Enter the full name of this id tag :
Server Database

Is Server Database used for client (c), server (s), or both (b) :
s

Enter the value of the sub directory path (the path that
gets appended to the path the user will enter) for Server Database
(e.g. @Arch@, @HostName@.db)(Press return to skip) :
@HostName@.db

Enter the default installation path or a comma separated list of choices
 for Server Database (Press Return to skip):
/var/spool/Tivoli

Enter optional environment variables needed
(e.g. @Env@)(Press return to skip) :

Enter id to chain from. (e.g. BIN, LIB, ...)
(Press return to skip) :

Creating the install packet/script entries for Server Database.

Select install packet (f) or script (s) (Press Return to continue) : f

First we need to know whether to create a install packet for
each each interptype or just one install packet.

Enter a '+' to indicate a per interptype install packet
or enter the desired contents for the arch field :
generic
```

```
Enter the path to the files to put in the install packet :
/:/project/distmon/dummy

Enter the path to a before script (Type Enter to skip) :
/:/project/distmon/install/ITSOMON_scripts/ALIDB_after_unix.sh

Enter the name of a file (a perl regex) to look for to decide if this por
is installed.  You can also add to the disk space requirements for insta
this package by adding a ; followed by a number after the file to look f
For example: fileB;1024 would specify to look for a file named fileB and
a check for an additional 1024K of disk space (Type Enter to skip) :

Warning: after script /:/project/distmon/install/ITSOMON_scripts/ALIDB_af

Creating the install packet/script entries for Server Database.

Select install packet (f) or script (s) (Press Return to continue) :

Enter an id tag (e.g. BIN, LIB, DB...).
This will be the tag that your install packets will
reference (Return to continue) :

Enter the flag for the gui item (e.g. L, G, S..).
reference (Return to continue) :

The logfile for this session is in /tmp/wcrtindex.log31276. You can edit
it and use the modified file as STDIN when you re-run this script.
```

Now the files ITSOMON.IND and ITSOMON.XREF have been created. A CFG directory containing two files has also been created:.

```
FILE2000.CFG
FILE2001.CFG
```

The file ITSOMON.XREF should contain:

```
2000;.;CFG/FILE2000.CFG;/:/project/distmon/code/SentryMonitors
2001;.;CFG/FILE2001.CFG;/:/project/distmon/dummy
```

Probably you will have to correct the syntax manually because it contains ":" instead of ";". We also had to change absolute paths to relative paths to get the right XREF format. Further, we had to add additional lines in the IND file to perform the prerequisite dependency checking.

Note

It is useful to compare your IND, XREF and CFG files with the contents of the Tivoli CD-ROM. It can help you to find invalid formats, also to add more variables if you want to. For example, in the after script, you can add the variables that are defined in the index file.

The following is the listing of our ITSOMON.IND file:

```
ITSOMON:Description:TME 10 Distributed Monitoring ITSO
Collection 1:
ITSOMON:id:BIN:Binaries:server:generic/
SentryMonitors:default=/usr/local/Tivoli/bin&colonThisDir=@BIN@;
ThisHost=@HostName@;ThisPkg=BIN;:
ITSOMON:fp:BIN:generic:tmon.col:122
:500
ITSOMON:id:ALIDB:Server Database:server:
@HostName.db:default=/var/spool/Tivoli:
ThisDir=@ALIDB@;ThisHost=@HostName@;ThisPkg=ALIDB;:
ITSOMON:fp:ALIDB:generic::2:501
ITSOMON:depends:SEN_36
```

Figure 188. Listing of ITSOMON.IND

You can see each line must begin with the keyword "ITSOMON". Then we define the names, location and variables for the Binaries and Server Database parts. The last line is the dependency checking.

The following is a listing of our FILE2000.CFG file:

```
#*TFP-v2.01
#version=ITSOMON_BIN_generic
do_compress=y
do_checksum=y
stop_on_error=n
descend_dirs=y
follow_links=y
create_dirs=y
unix_default_dir_gid=bin
unix_default_file_gid=bin
unix_default_dir_uid=root
unix_default_file_uid=root
post_notice=n
#
# File List:
#
%
cpu.col
fsmond.col
mon8235.col
monvm.col
snapd.col
snapm.col
tmon.col
%
```

Figure 189. Listing of CFG/FILE2000.CFG

You see that the list of the files to copy to the Tivoli directory is defined here. No after or before script is run in the Binaries pass.

The following is the listing of our FILE2001.CFG file:

```
#*TFP-v2.01
#version=ITSOMON_ALIDB_generic
do_compress=y
do_checksum=y
stop_on_error=n
descend_dirs=y
follow_links=y
create_dirs=y
unix_default_dir_gid=bin
unix_default_file_gid=bin
unix_default_dir_uid=root
unix_default_file_uid=root
post_notice=n
unix_after_prog_from_src=y
unix_after_prog_path=/:/project/distmon/tme_install/ITSOMON_scripts/ALIDB
_after_unix.sh
#
# File List:
#
%
%
```

Figure 190. Listing of CFG/FILE2001.CFG

You can see there is no file to copy in the Database Server pass. The after script is defined in this file. At this step the directory structure is ready to create the image.

4.4.1 Creating the Image

The ADE command wcrtimage is used to create the packet files .PKT and also a file CONTENTS.LST.

```
wcrtimage ITSOMON.XREF

using a cdrom directory of .
Making ./FILE2000.PKT from ./CFG/FILE2000.CFG
        using /:/project/distmon/code as source...

Making ./FILE2001.PKT from ./CFG/FILE2001.CFG
        using /:/project/distmon/dummy as source...

adding ITSOMON:TME 10 Distributed Monitoring ITSO Collection1
 to ./CONTENTS.LST
```

Figure 191. Output of wcrtimage Command

The .PKT files and the CONTENTS.LST file are created in the current installation directory. The CONTENTS.LST file contains the description of our product:

```
ITSOMON:TME 10 Distributed Monitoring ITSO Collection 1
```

You can check the contents of the .PKT files with the sapack command. This command is located under $BINDIR/TAS/INSTALL.

```
sapack -up FILE2000.PKT

100664  1   0/  2   21289  Mar 03 15:04:39 1998  cpu.col
100664  1   0/  2    8461  Mar 03 15:04:41 1998  tmon.col
100664  1   0/  2   13007  Mar 03 15:04:40 1998  monvm.col
100664  1   0/  2   11945  Mar 03 15:04:40 1998  mon8235.col
```

The file FILE2001.PKT does not contain sapack format, as it is only used to perform an after script operation.

4.5 Testing the Installation

Once our image resides on a CD-ROM-like directory, we can try to install our custom product from a Tivoli administrator desktop. From the Tivoli Desktop, select **Desktop** from the menu bar. Then select **Install -> Install Product** from the pull-down menu as shown below:

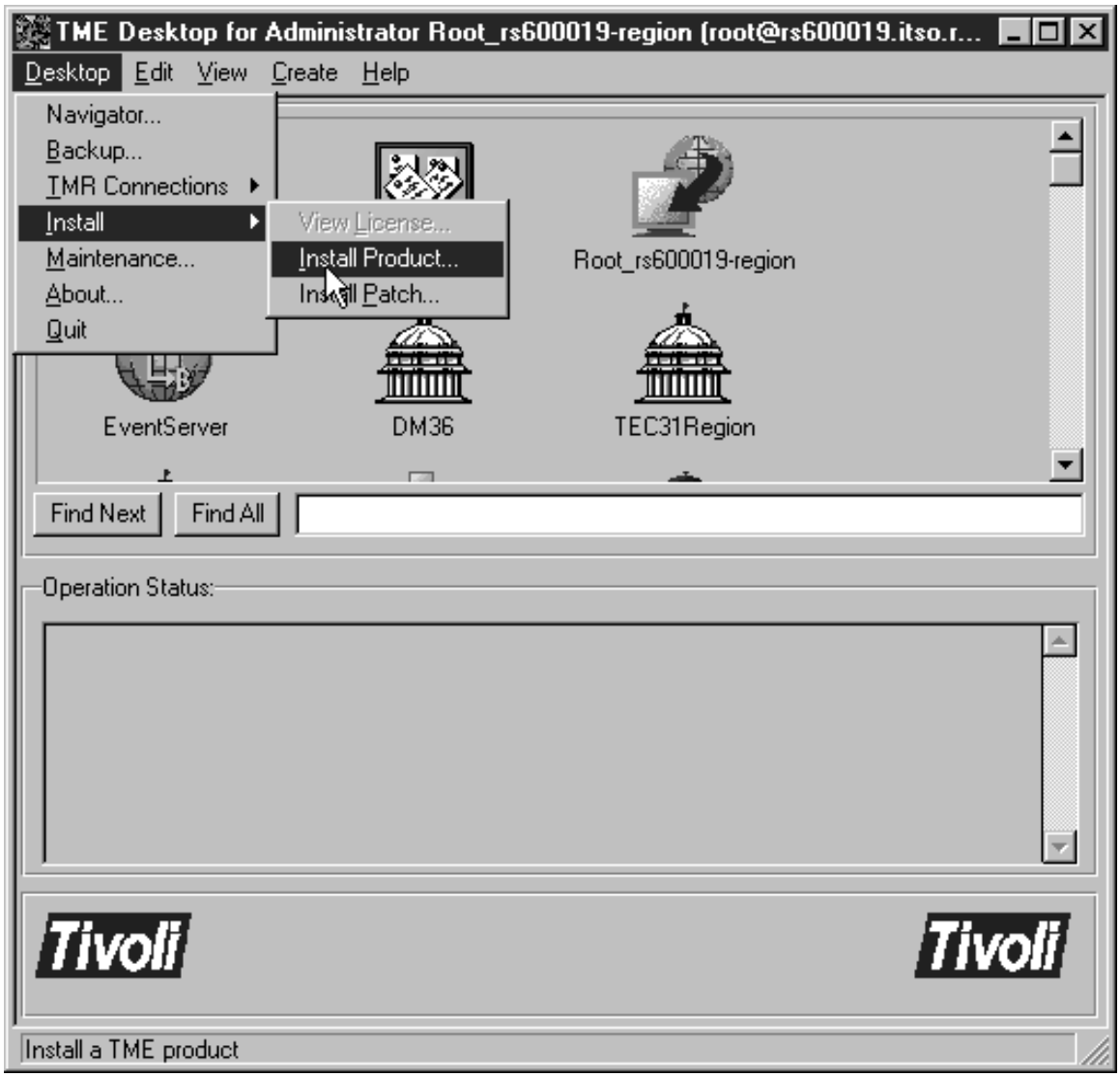


Figure 192. Install Product Pull-Down Menu

The File Browser window will appear.

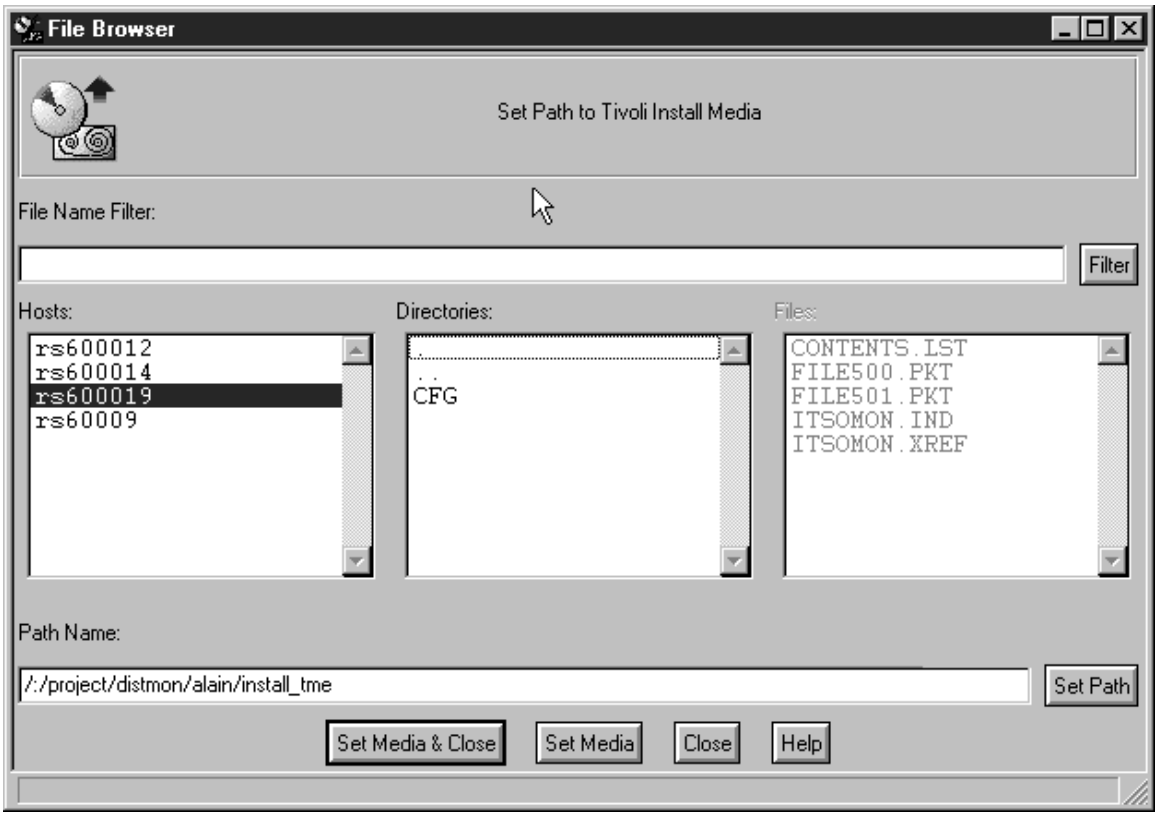


Figure 193. File Browser Window

Click on the **Set Media** button to set the media to the correct directory. Once the media is correct, the Install Product window should appear.

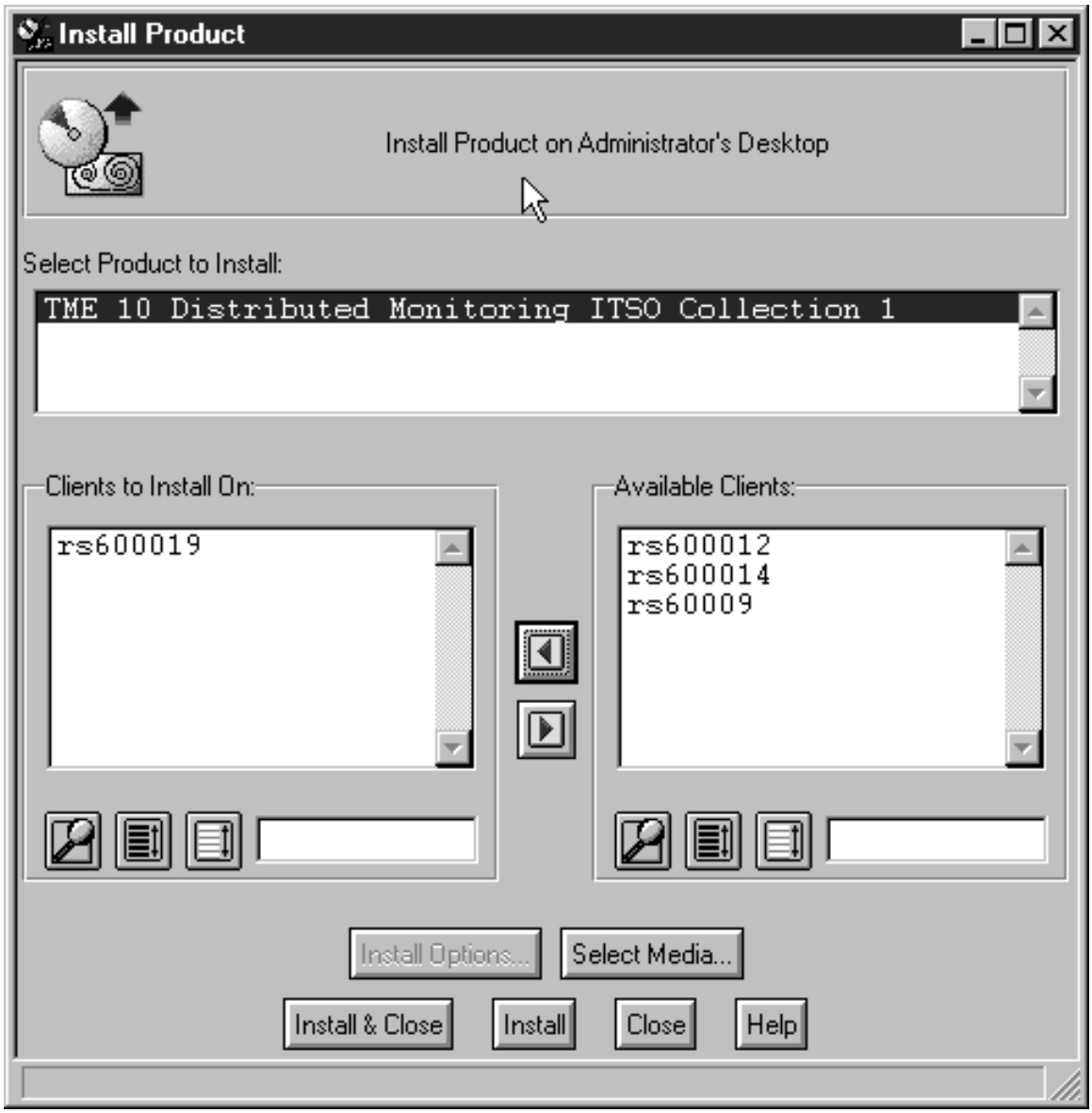


Figure 194. Install Product Window

Click on **TME10 Distributed Monitoring ITSO Collection 1** to select the product to install. Click on **Install & Close** and you will be prompted to confirm that you want to perform an installation.

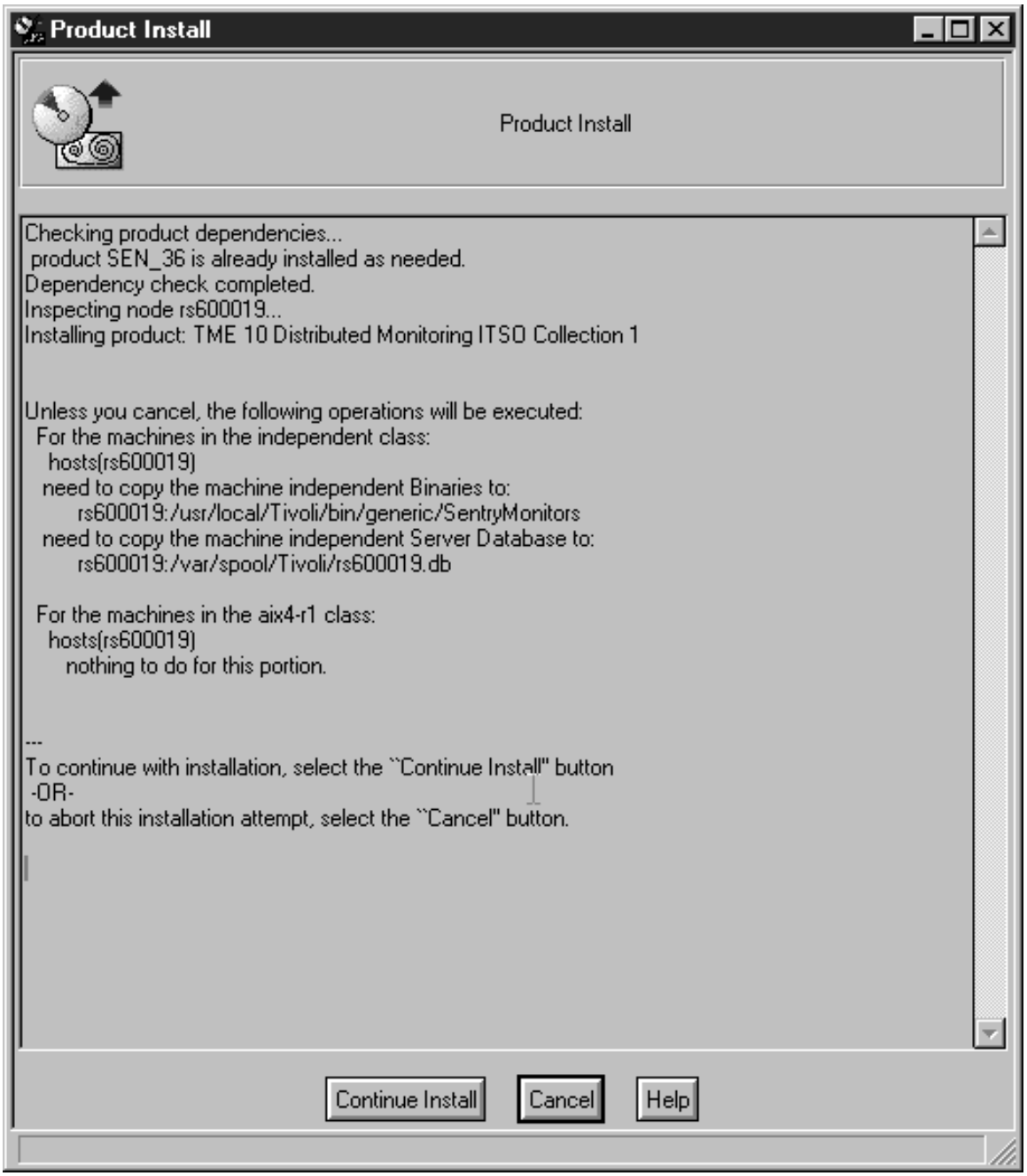


Figure 195. Product Install Window

Click on **Continue Install** and the installation will proceed. We see in this window that the two passes are executed in sequence:

- Distributing machine-independent Binaries
- Distributing machine-independent Server Database

Then the product is registered in the Tivoli name registry.

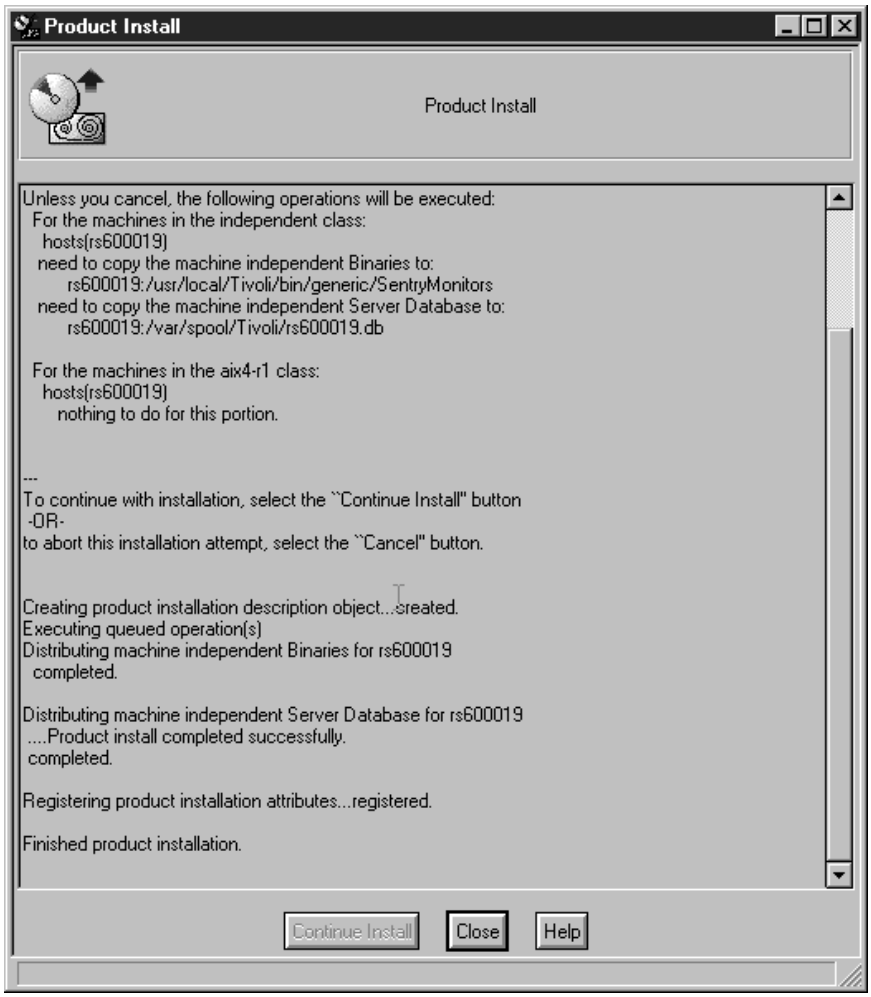


Figure 196. Product Install Window

At this step the installation has been performed successfully. Click on **Close** to close the window. You can check the product installation by running the command:

```
wlsinst -a
```

This command will return the list of all the TME 10 products installed on your system:

```
*-------------------------------------------------------------
                        Product List
*-------------------------------------------------------------

TME 10 Framework
TME 10 ADE, Version 3.2
TME 10 ADE Postscript Documentation, Version 3.2
TME 10 Distributed Monitoring ITSO Collection 1
TME 10 Enterprise Console 3.1 Rule Builder
Tivoli Distributed Monitoring 3.6
TME 10 Enterprise Console 3.1
TME 10 Enterprise Console 3.1 Server
TME 10 Distributed Monitoring Universal Monitors
TME 10 Distributed Monitoring Unix Monitors
```

You see that our new product has been added in the list.

4.6 Using a Custom Notice Group

Once the newly created product is installed on our TMR server, we want our custom monitors to report information to a specific notice group. We have to create a custom notice group. This group will be named "ITSO Group" and our custom monitors will post their notices to that specific group.

4.6.1 Creating the Notice Group

There is currently no Tivoli command to create a custom notice group, so we have to use an IDL call. Here are the commands to run:

```
NTFGM=`wlookup -r Classes TMF_Notice`
idlcall -T top $NTFGM
TMF_Notice::NoticeManager::create_notice_group
'"ITSO Group" 186'
```

The first command gets the instance manager for notice groups and assigns it to a variable NTFGM. The second command is an idl command that invokes the method "create_ notice_group" on the OID $NTFGM. This method requires two arguments:

1. The name of the new group. Here we have chosen "ITSO Group".
2. The expiration time in hours. The standard default is 1 week (186 hours).

Note

The back quotes are required for IDL to treat the argument as a string

You can check that the new notice group is in the name registry with a wlookup command:

```
wlookup -r TMF_Notice "ITSO Group"
1237321695.1.743#TMF_Notice::Group#
```

4.6.2 Subscribing to the Notice Group

After the creation of our custom Notice Group, no Tivoli Administrator (including Root) is initially subscribed to this group. To subscribe Root to the new group, double-click on the **Administrators** icon on the TME Desktop. Then select **Edit Notice Group Subscriptions...** from the Root Administrator pull-down menu, as shown in Figure 197 on page 208.

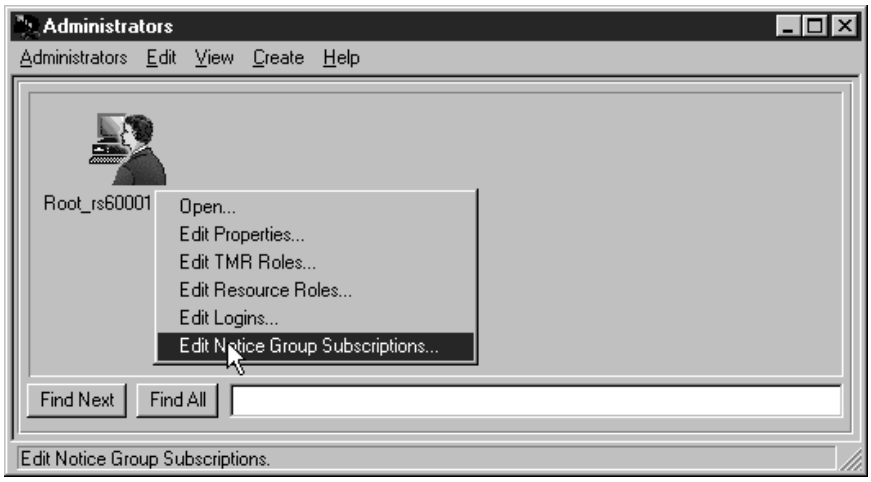


Figure 197. Edit Notice Group Subscriptions Pull-down Menu

The Set Notice Groups window appears:

Figure 198. Set Notice Groups Window

Now Click on **ITSO Group** from the Available Notice Groups and click on the left arrow button. The new notice group is moved from the Available Notice Groups to the Current Notice Groups scrolling list. You can check the new Notice Group by double-clicking on the **Notices** icon on the Tivoli desktop. Our new group appears in the Read Notices Window.

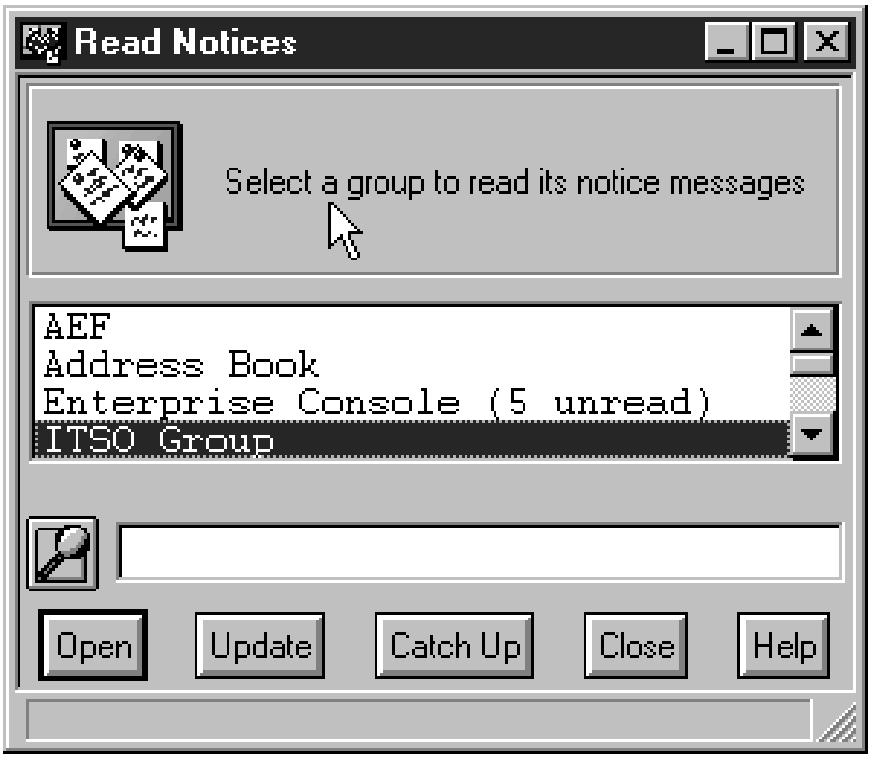


Figure 199. Read Notices Window

At this step our custom notice group is fully integrated in the TME 10 Framework environment.

4.6.3 Posting a Notice

Now we want our custom monitor to post its notices to our new group. The first thing to do is to modify the .csl files. These definition files contain a line like the following:

```
NoticeGroup = "Sentry ";
```

This line means that by default all the notices from this collection will be posted to the Sentry notice group. We simply replace this line, using our favorite editor, by:

```
NoticeGroup = "ITSO Group";
```

Then it is necessary to re-compile the .csl files using mcsl -P, and also to re-install the collection with mcsl -i -R. After that, when you add a monitor from one of the custom collections, you will see the default Send Tivoli Notice field is now ITSO Group in the Edit Monitor window:

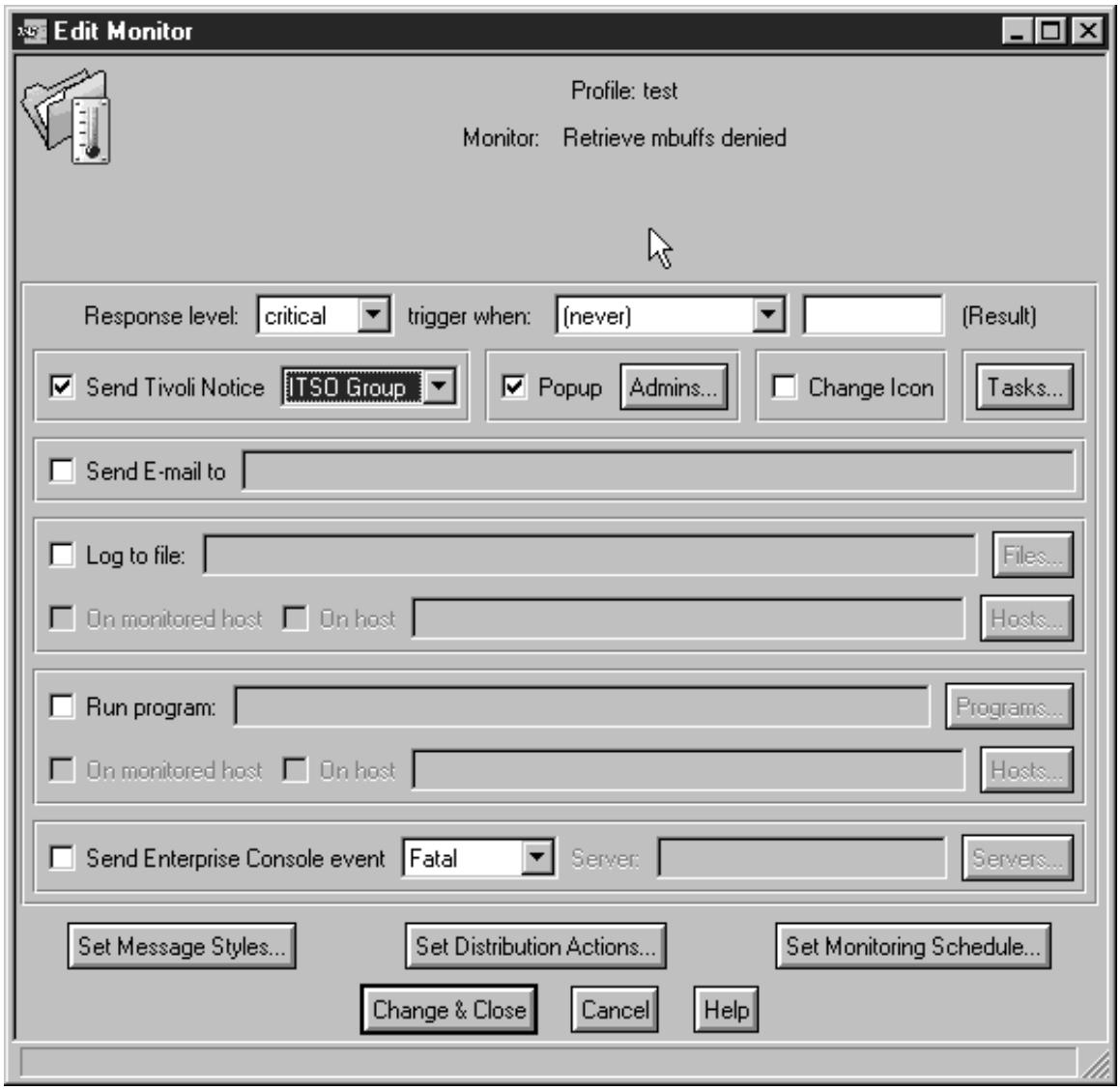


Figure 200. Edit Monitor Window

When our custom monitor, for example Custom Ping, sends an alert, a notice will appear in the ITSO Group.

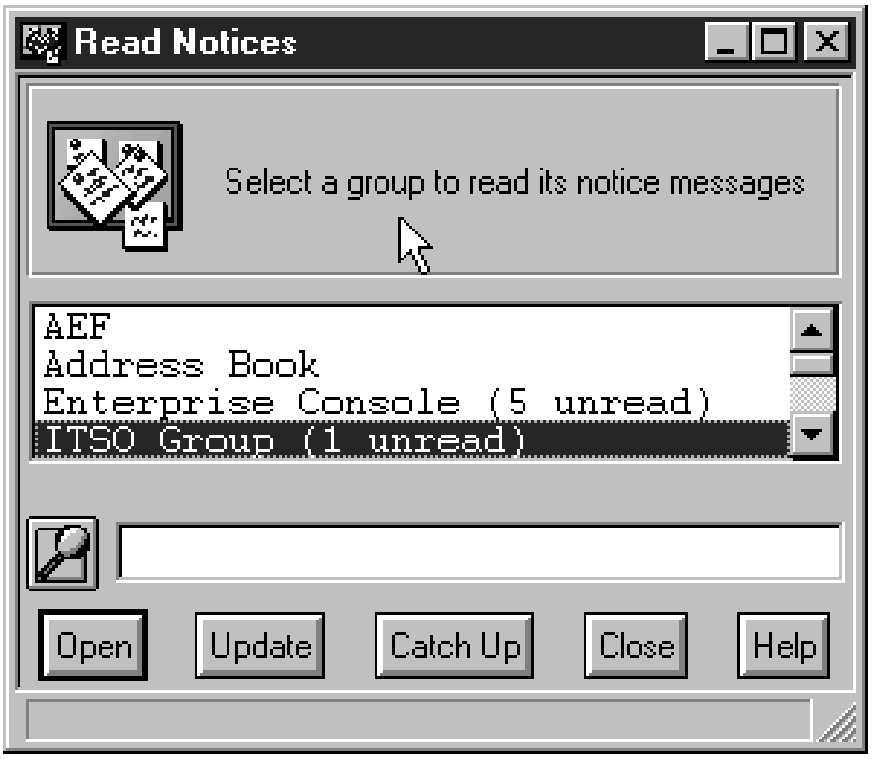


Figure 201. Read Notice Window

Click on **ITSO Group** and press the **Open** button to read the notice. You can read the message coming from our Custom Ping monitor:

Figure 202. Notice Group Messages Window

Chapter 5. Troubleshooting

In this chapter we describe some possible trouble you can encounter during the creation of custom monitors. We will briefly describe some of the problems we encountered, and the way we fixed them.

5.1 Compiling the Message Catalog

The following will cause compilation errors of a message catalog:

1. ADE not installed

   ```
   gencmsg /:/project/distmon/alain/test/testM.msg
   ksh:gencmsg:  not found.
   ```

   To fix this, make sure that the ADE product is installed with wlsinst -a

2. Missing or Invalid .msg file.

```
gencmsg /:/project/distmon/alain/test/testM.msg
gencmsg: cannot read /:/project/distmon/alain/test/testM.msg
```

To determine the cause, do the following:

- Check for the existence of file "testM.msg".
- Look for syntax errors.

5.2 Compiling the Monitor Definition File

The compilation of your .csl will be successful only if the syntax is correct and if the import reference is resolved. The most common compilation error we encountered was the following:

```
mcsl -P cpp -x testM.col testM.csl -lang-c++ -nostinc -undef
cpp: cannot retrieve message number 1501-232
cpp: cannot retrieve message number 1501-232
cpp: cannot retrieve message number 1501-232
testM.csl, line 17:       Description = (testM_MonSrc_Descr1);
Message error: missing comma
```

There are a few things to check:

- Check the existence of files testM.msg, testM.h, testM.c, testM.dsl in the current directory (Have you run gencmsg?).
- Check the testM.msg file and look for a line containing:

  ```
  $ key=MonSrc_Descr1
  ```

  If this reference doesn't exist, the compilation will fail.

Note

The cpp error `cannot retrieve message number 1501-232` is a non-fatal error. Probably this error is coming from the AIX 3.2.5 preprocessor we are using on an AIX 4.2.0 system.

To compile your collection file, you need a C pre-processor called cpp. There are different versions of cpp for UNIX. With some versions you can get an error like the following:

```
mcsl -P cpp -x testM.col testM.csl -lang-c++ -nostinc -undef

testM.csl, line 1: ---CPP---
Monitoring capability collection error: expected collection header
```

Ensure you are using the correct version of cpp for your platform type. In our examples we use the AIX Version 3.2.5 C Language pre-processor. If you try with a cpp that comes with X11 you should encounter this kind of error.

It is also important to have access to the necessary include files:

```
mcsl -P cpp -x alainM.col alainM.csl -lang-c++ -nostinc -undef
cpp: cannot retrieve message number 1501-232
cpp: cannot retrieve message number 1501-232
cpp: cannot retrieve message number 1501-232
./operators.csl, line 6:         { Label = (Sentry2_0_OpNever); };
Message error: missing comma
```

Ensure you have all the include files defined in your .csl file, in this order:

```
#include "Sentry2_0.dsl"
#include "operators.csl"
#include "formats.csl
```

These include files are supposed to reside in our current working directory.

Note

During the creation of MCSL scripts, we had problems compiling csl files whose names contain non-alphanumerical characters. We recommend that you do not use characters like "_" or "-" in file names.

For example, if you change all the occurrences of userdelay to user_delay (both in file names and in the .csl file), you should have this kind of error:

```
mcsl -P cpp -x user_delay.col user_delay.csl -lang-c++ -nostinc -undef
cpp: cannot retrieve message number 1501-232
cpp: cannot retrieve message number 1501-232
cpp: cannot retrieve message number 1501-232
    : (S) No message text for 00610006
user_delay.csl, line 9:         HelpMessage = (user_delay_About_MonCol);
Message error: missing comma
```

The error message `missing comma` is a generic error message. It doesn't actually mean that a comma is missing; it is a compilation error. If you use "userdelay", you will not have this error any more.

Another thing to check is the path to the script or the program you reference in your import definition. If you don't use an absolute path, the script or the program must reside on the current directory. If you use an absolute path, ensure this path is correct. Otherwise, you will get an error like this:

```
mcsl -P cpp -x testM.col testM.csl -lang-c++ -nostinc -undef
cpp: cannot retrieve message number 1501-232
cpp: cannot retrieve message number 1501-232
cpp: cannot retrieve message number 1501-232
testM.csl, line 28:       ;
Implementation error: cannot open import file
```

We must check the import definition instruction in our testM.csl file:

```
Import "/tmp/test.sh"
```

The shell script test.sh doesn't exist under /tmp. We copy this script file to /tmp and the definition is resolved.

5.3 Custom Script Monitors Not Working

If the custom scripts monitors do not work, there are a few things to check:

- The script must print its result to stdout.
- It must exit with an exit code 0.
- It must start with a line #!/bin/sh or equivalent.
- It must be executable by the user ID that the monitor is running under. By default, this is nobody.

You can check this by editing your monitor properties and selecting **Edit** from the menu bar and then **Set User & Group ID** from the pull-down menu.

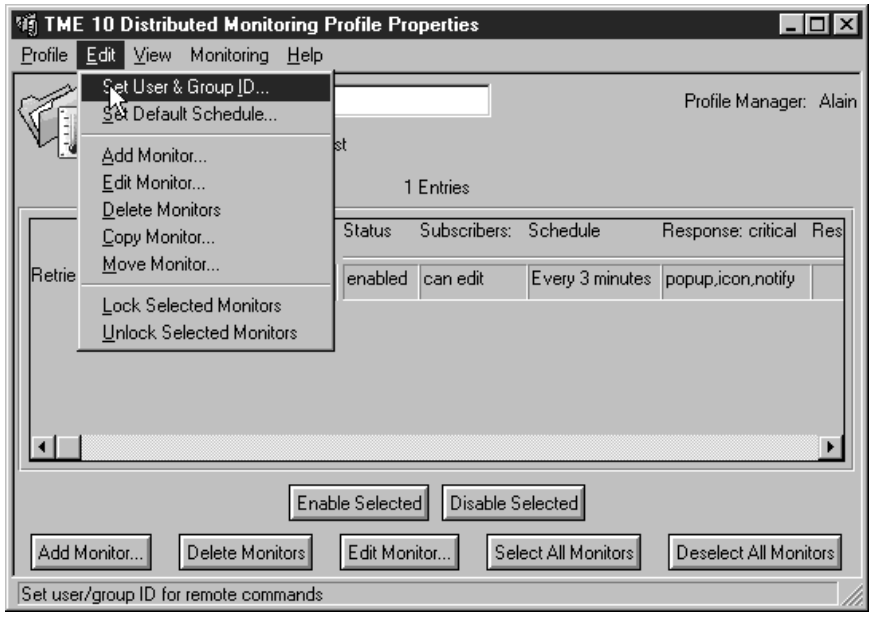


Figure 203. TME 10 Distributed Monitoring Profile Properties Window

You see that by default the UID/GID is nobody/nobody.

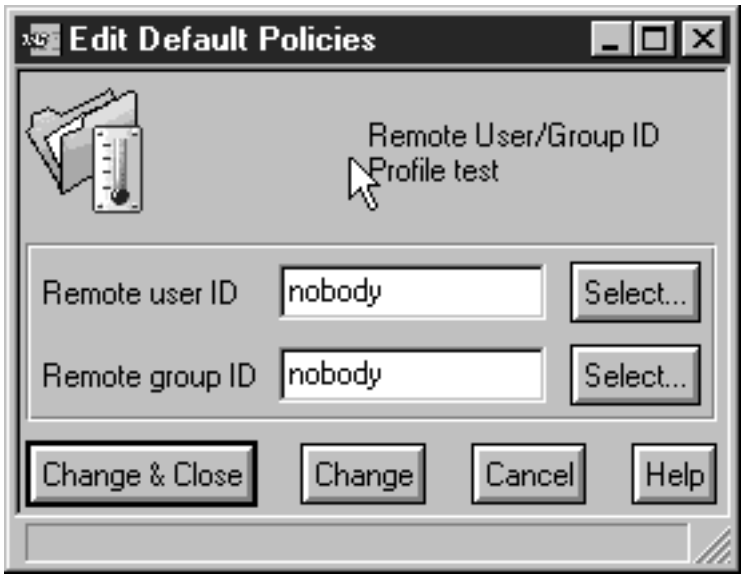


Figure 204. Edit Default Policies Window

The first thing to do is to switch to user nobody and try to execute the shell script. If you use some Tivoli commands in the script or in the C program, you must set an UID/GID that is a valid login name to a Tivoli Administrator, with the right authority. If this is not the case, you might get "insufficient authorization" errors.

It is also always useful to set the response level of your monitor to `always` and to log to a file. This log may contain more detailed information than a Tivoli notice. You can also add a debug section in the script, using an environment variable, for example DEBUG. If the script has DEBUG=1, it can log more additional information to a file. You can also use Distributed Monitoring commands that may be useful in case a monitor is not working as expected:

- `wlsmon` lists Sentry profiles.
- `wdumpsnt` lists the Sentry definition.
- `wloadsnt` loads the Sentry definition.
- `wsetmon` enables or disables a Sentry monitor.
- `wclreng` clears the Sentry engine.
- `wdelsnt` deletes a Sentry profile from a client engine.
- `wdelprb` deletes a specified Sentry monitor from a client engine.
- `wrunprb` runs a specified Sentry monitor from a client engine.
- `wlseng` lists the contents of the Sentry engine.

5.4 Deleting a Collection

If you want to get rid of a collection, it is necessary to delete an object in the Tivoli database. The collections are defined in the Tivoli name registry, under the resource name "MonitoringCapabilityCollection". It is a good idea, before deleting an object, to make a backup of your Tivoli Database. After that, you run the command:

```
wlookup -ar MonitoringCapabilityCollection
```

and you note the name of the object you want to delete among the list of all collections. Then you run the command:

```
wdel /Library/MonitoringCapabilityCollection/<name-of-custom-coll>
```

and finally run a wchkdb -u to clean up the database, if needed.

You can check if the collection has been removed by typing:

```
mcsl -c
```

This command should return something like this:

```
Collection Universal is license-protected (products: "Universal",
"Tivoli/sentry")
Collection Unix_Sentry is license-protected (products: "Unix_Sentry",
"Tivoli/sentry")
```

Then restart the Sentry engine with the mcsl -R command.

Appendix A. Special Notices

This publication is intended to help IT Specialists to understand how to create custom monitors for Tivoli Distributed Monitoring. The information in this publication is not intended as the specification of any programming interfaces that are provided by Tivoli Distributed Monitoring. See the PUBLICATIONS section of the IBM Programming Announcement for Tivoli 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.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

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:

AIX	DB2
IBM	NetView
OS/2	SystemView

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

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

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

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

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

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

Appendix B. Related Publications

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

B.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 223.

- *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
- *Migrating from Systems Monitor for AIX to TME 10 Distributed Monitoring*, SG24-4936
- *TME 10 Inventory Cookbook*, SG24-2120
- *TME 10 Inventory 3.2 New Functions and Database Support*, SG24-2135
- *Understanding Tivoli's TME 3.0 and TME 10*, SG24-4948
- *TME 10 Framework Version 3.2: An Introduction to the Lightweight Client Framework*, SG24-2025
- *Tivoli Software Distribution 3.6: Unleashing the Power of TME and LCF*, SG24-2045 (Available at a later date.)

B.2 Redbooks on CD-ROMs

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

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at `http://www.redbooks.ibm.com/`.

How IBM Employees Can Get ITSO Redbooks

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

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type `GOPHER.WTSCPOK.ITSO.IBM.COM`
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type the following command:

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

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1998
  ```

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

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

- **Redbooks Web Site on the World Wide Web**

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

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

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

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to `announce@webster.ibmlink.ibm.com` with the keyword `subscribe` in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

Redpieces

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

How Customers Can Get ITSO Redbooks

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

- **Online Orders** — send orders to:

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

- **Telephone orders**

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

- **Mail Orders** — send orders to:

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

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

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

- **Direct Services** - send note to `softwareshop@vnet.ibm.com`
- **On the World Wide Web**

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

- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to `announce@webster.ibmlink.ibm.com` with the keyword `subscribe` in the body of the note (leave the subject line blank).

Redpieces

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

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

□ Invoice to customer number

□ Credit card number

Credit card expiration date Card issued to Signature

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

Index

Special Characters

Numerics

A

B

C

D

N

O

P

Q

R

S

T

U

V

W

X

ITSO Redbook Evaluation

Creating Custom Monitors for Tivoli Distributed Monitoring
SG24-5211-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
□Customer □Business Partner □Independent Software Vendor □IBM employee
□None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____

_____

_____

_____

_____

SG24-5211-00
Printed in the U.S.A.