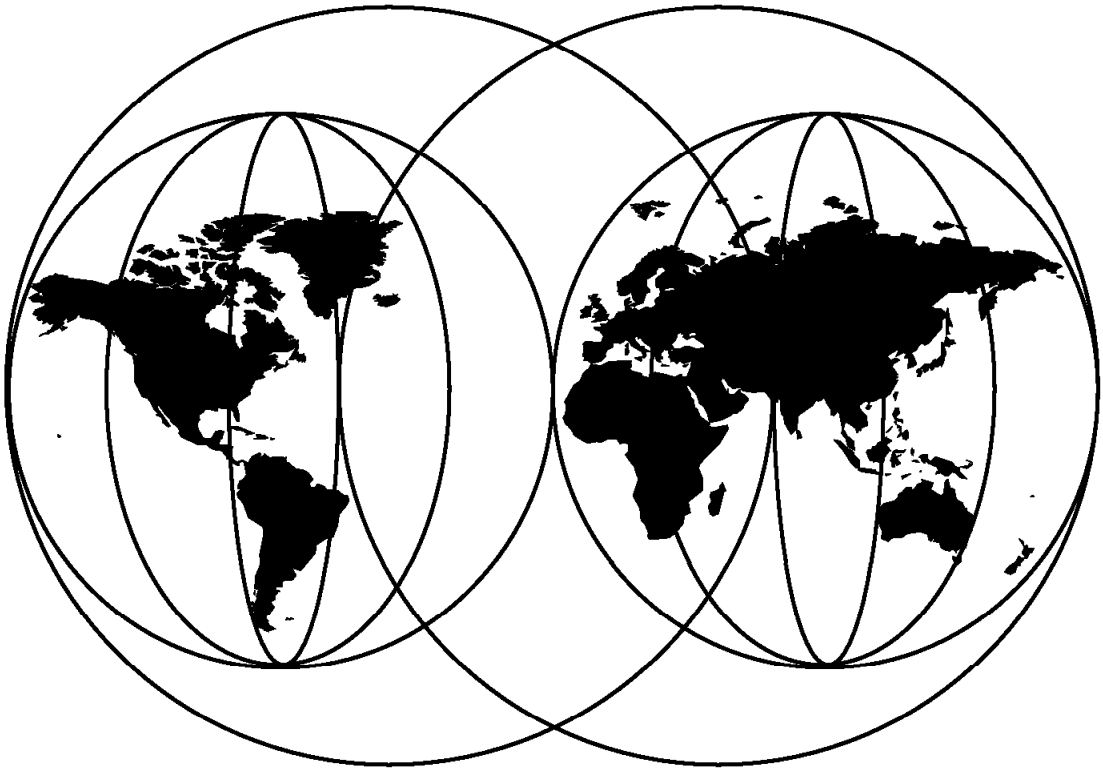
Using Tivoli′s ARM Response Time Agents

Fergus Stewart, Mohamed Shoaib Dawood, Inmaculada Doblas, Gary Griffith, Diane Nelson



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-2124-00

International Technical Support Organization

SG24-2124-00

Using Tivoli′s ARM Response Time Agents

July 1998

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix H, “Special Notices” on page 147.

First Edition (July 1998)

This edition applies to TME 10 Distributed Monitoring 3.5, and to the ARM agents for use with the operating systems that support them.

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

Preface

This redbook explains what Tivoli's ARM Agents are, and what they can do.

It shows you how to install and set up the Agents, and how to display the data they collect in a Web browser and with Reporter.

There is a discussion of how to make the best use of the Agents in a production environment.

The book also explains how to instrument your applications for the ARM API, so that you can measure performance, account for their use, and monitor availability.

This redbook should be especially helpful if:

- You want to measure response times for your application users.
- You know that you have response time problems, and you want to isolate the cause.
- You are wondering what ARM is all about.
- You want to know what the ARM Agents can do for you.
- You have Distributed Monitoring and want to know how to install and set up the ARM Agents.
- You want to know how to make the best possible use of the ARM Agents.

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.

Fergus Stewart is an Advisory International Technical Support Specialist at the Systems Management and Networking ITSO Center, Raleigh. He writes extensively and teaches IBM classes worldwide on network performance. Before joining the ITSO three years ago, Fergus worked in IBM UK. He can be contacted at *fergus@us.ibm.com*.

Mohamed Shoaib Dawood is an IT Specialist in IBM Global Services in South Africa, where he has been working since January 1997. He holds a BSc. degree in Computer Science from the University of Durban Westville. Mohamed is involved in a project to implement Tivoli management for PC servers and desktops in IBM and customer sites.

Inmaculada Doblas is a Systems Engineer in IBM Spain. She has a computer science degree from the Universidad de Malaga. Inmaculada has been working for IBM for the last three years; her skills are in the areas of C programming, AIX, and network administration.

Gary Griffith is a Network Systems Analyst at the Boeing Company in Seattle, Washington. He will complete a BAcs degree at Seattle Pacific University in 1998. His areas of expertise include VTAM/NCP, Windows NT Server, IMS and hardware encryption. At Boeing he is the project manager for the RS/6000 SP gateway, and is currently involved in implementing Tivoli NPM, Communications Server for AIX and Host on-Demand.

Diane Nelson is a Software Systems Analyst at Aid Association for Lutherans, in Appleton, Wisconsin. She holds a BS degree in Computer Information Systems/Business from Northern Michigan University, and a Masters from Silver Lake College in Management and Organizational Behavior. Her areas of expertise include AIX, Distributed Monitoring and Performance Management.

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

Arne Olsson
Margaret Ticknor
Stefan Uelpenich
Systems Management and Networking ITSO Center, Raleigh.

Astrid Burnette
Mark Johnson
Tivoli Systems, Austin

Silvia Bellucci
Paolo Papi
Andrea Saracini
Marco Sebastiani
Pino Venturella
Tivoli Systems, Rome

Herbert Arguello
Tivoli Systems, San Francisco

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 159 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 to Application Response Measurement (ARM)

The Application Response Measurement (ARM) API is a set of standard API calls, agreed upon by a number of prominent IT companies, that allow an application-centric focus on performance.

The API is designed to allow you to monitor the performance of any application. The most likely use of the application monitoring capability will be to measure response time, but it can also be used to record application availability and account for application usage.

Previous techniques for performance measurement have taken one of the following approaches:

- Monitor the network that connects a client to a server, and record the interval from when a request leaves the client until a response arrives at the client.

  This has the following weaknesses:

  - It does not measure any processing performed on the client machine.
  - It expects only one outbound flow, and one complimentary inbound flow to the client.

- Use a program that generates a typical client workload, and make time measurements within the program.

  This has the following weaknesses:

  - It does not measure actual user work, and if users are aware of this, they will not believe it.
  - Real user transactions inevitably differ from the model workload that the program uses.

- Measure the utilization of all the hardware components involved in providing service to the user, and assume that if these values are at reasonable levels, the user response time must be acceptable.

  This has the following weaknesses:

  - It does not provide actual response time measurements that can be reported against a service level agreement.
  - Unhappy users who are experiencing extended response times are not likely to be pacified by a statement that the utilization of the server is at an acceptable level.

1.1 Instrumenting Applications for Response Time Measurement

In order to measure response times for an application, the application must be instrumented to the ARM API. This means that there must be ARM API calls in the application code.

You can see how the calls are used to measure response times in Figure 1 on page 2.

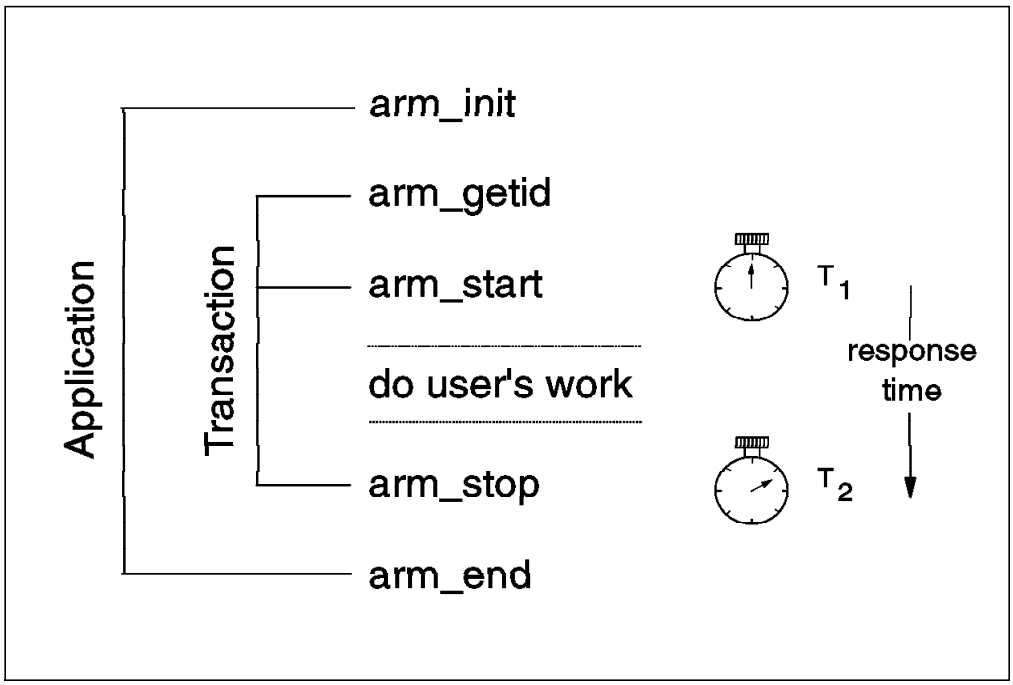


Figure 1. The Relationship Between ARM Calls

There are six ARM API calls, as follows:

- arm_init

  This is used to define an application to the response time agent.

- arm_getid

  This is used to define a transaction to the response time agent. A transaction is always a child of an application.

- arm_start

  This call is used to start the response time clock for the transaction.

- arm_update

  This call is optional. It can be used to send a heartbeat to the response time agent, while the transaction is running. You might want to code this call in a long-running transaction, to receive confirmations that it is still running.

- arm_stop

  The arm_stop call is used to stop the response time clock when a transaction completes.

- arm_end

  This call ends collection on the application. It is effectively the opposite of the arm_getid and arm_init calls.

You can see an example of the calls in A.1, “SIMPARM.C” on page 115.

The benefit of this approach is that you can place the calls that start and stop the response time clock, in exactly the parts of the application that you want to measure. This is done by defining individual applications and transactions within a program, and placing the ARM API calls at transaction start and transaction end.

Of course the disadvantage of this approach is that you may need to invest some programming effort in identifying the places in the application code at which you want to call the ARM API, and then placing the API calls there.

In this book we have used the instrumentation for Lotus Notes on Windows 95 and NT, which is more fully documented in *Measuring Lotus Notes Response Times with Tivoli's ARM Agents*, SG24-4787.

We plan to work on the instrumentation of other user applications and document these in future books. Please let us know (see comments welcome) of other applications that you would like to see instrumented.

1.2 Response Time Collection

Response time collection is performed by the following components:

- ARM Server Agent

  The ARM Server Agent is responsible for summarizing the collected response time data, and communicating with the ARM Manager and Reporter.

- ARM Client Agent

  The ARM Client Agent receives ARM API calls from the user application, calculates response times, and communicates with the ARM Server Agent. It must reside on the same machine as the user application.

The ARM Server Agent and the ARM Client Agent can run on the same machine, or on different machines.

As we saw in 1.1, "Instrumenting Applications for Response Time Measurement" on page 1, in order for an ARM agent to collect response times for an application, that application must be instrumented for ARM. What this means in practice is that the user application must make calls to the ARM API.

Only the ARM Client Agent needs to be installed on the end-user machine where response time measurements are being made.

1.3 ARM Agents Available Today

The ARM agents are available with Distributed Monitoring 3.6.

As we saw in 1.2, "Response Time Collection," the ARM Client Agent must be running on the machine where response time measurements are to be made. ARM agents are currently available to measure response times on the following operating system platforms:

- Windows 95
- Windows NT
- AIX
- HP-UX
- Sun Solaris

Windows 16-bit Operating Systems

According to the Distributed Monitoring documentation, Windows 3.1 and Windows for Workgroups are not supported as ARM clients. We made several attempts to get the Windows agent working on Windows 3.1, but we were not successful.

The ARM Client Agents communicate with ARM server agents using TCP/IP, as shown in Figure 2.

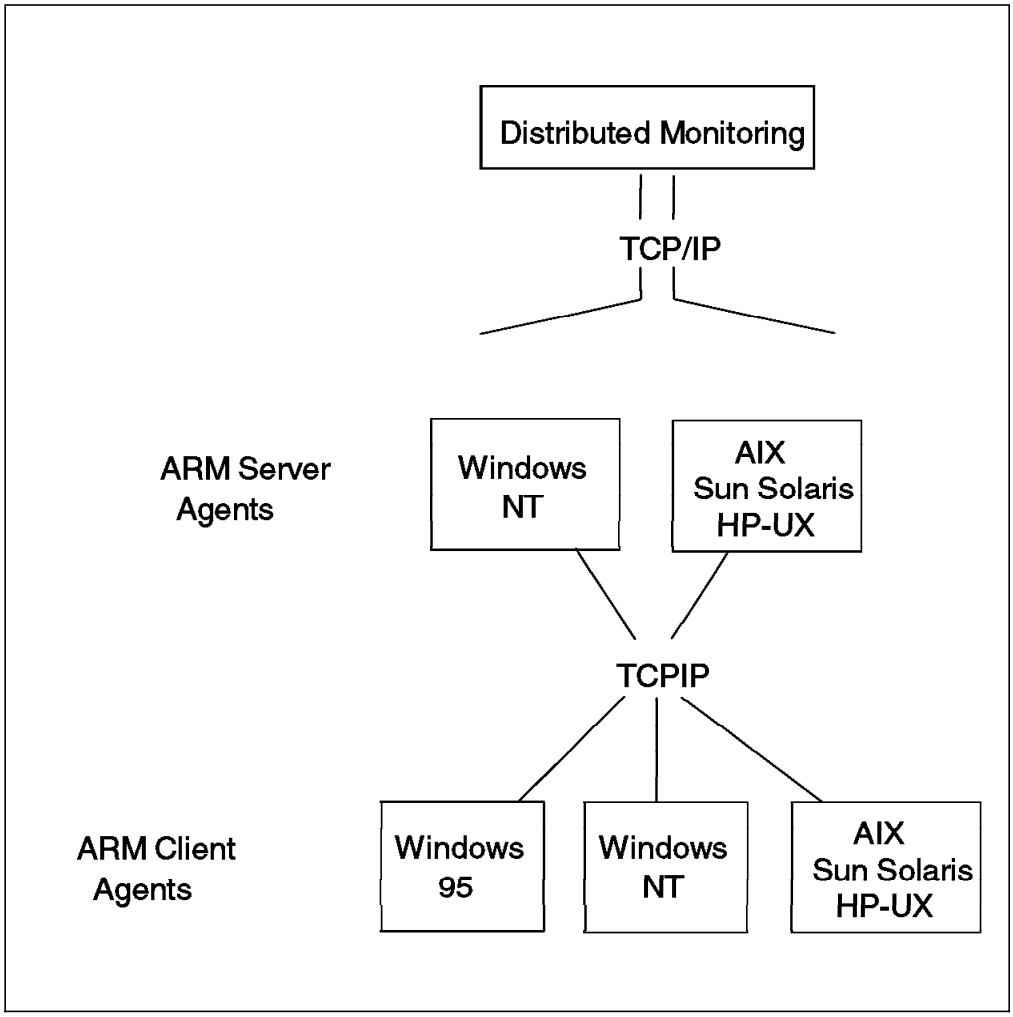


Figure 2. ARM Server to ARM Client Communications

1.4 ARM in Tivoli Systems Management

As a part of Distributed Monitoring, Tivoli's implementation of ARM takes full advantage of the functions provided by the Tivoli framework.

To use ARM you will need the following products:

- Tivoli Framework Version 3.2
- Distributed Monitoring Version 3.5

To get the most from an ARM implementation, it can be integrated with the full suite of Tivoli products, as shown in Figure 3 on page 5.

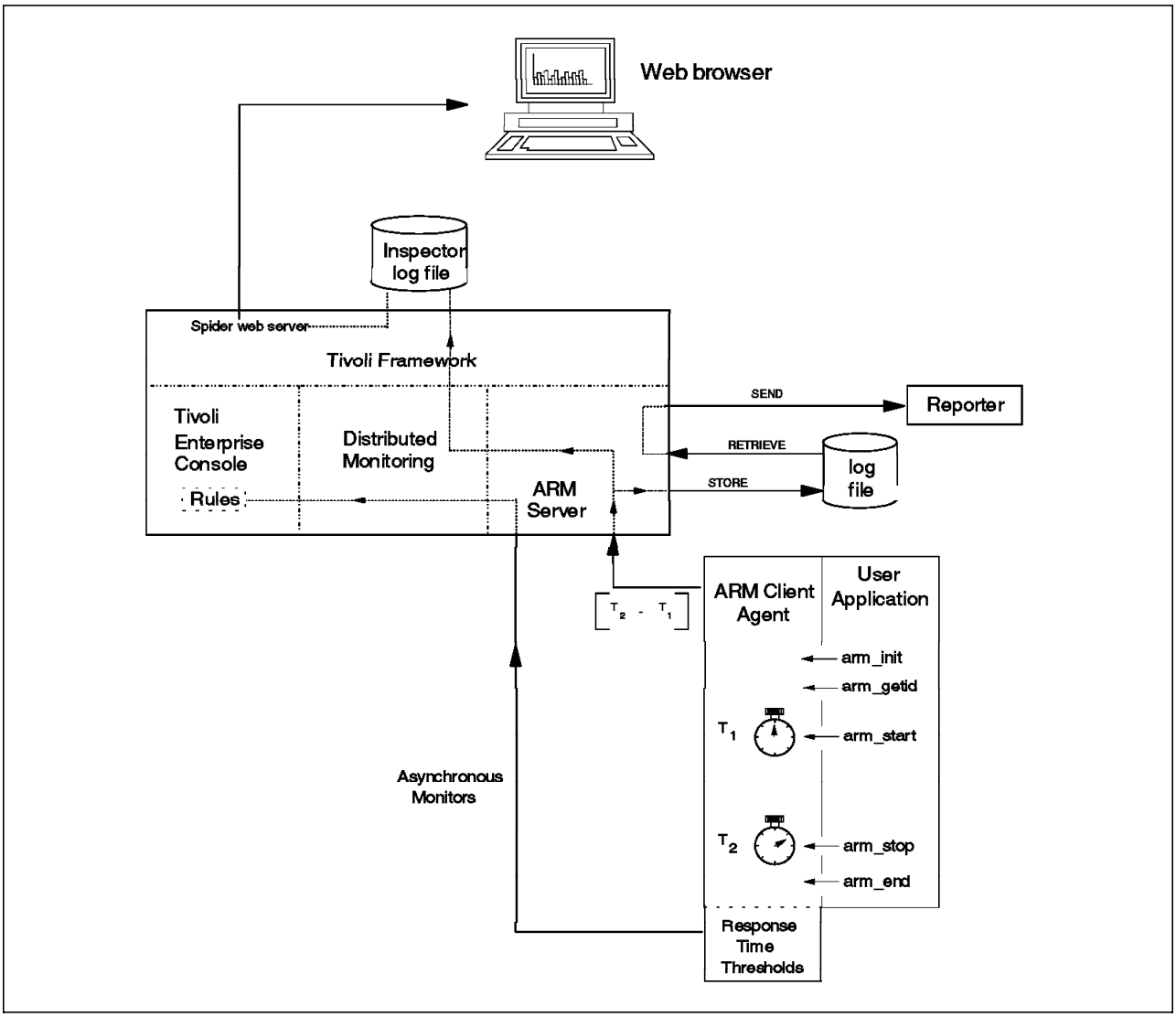


Figure 3. The ARM Products Infrastructure

Note that Distributed Monitoring, Reporter, TEC, the ARM Server Agent and the ARM Client Agent can be on the same machine, or on different machines.

Chapter 2. Setting Up an ARM Environment on Our TMR Server

We decided to start by installing all the ARM products on one machine that was running Windows NT Server 4.00.1381 (Service Pack 3).

The steps to set up an ARM environment on a single machine are as follows:

1. Install Tivoli Management Framework Version 3.2
2. Install Distributed Monitoring 3.5
3. Install the Distributed Monitoring ARM Agents and ARM Monitors
4. Create a Distributed Monitoring Proxy

We will not cover the installation of the Tivoli Management Framework Version 3.2 here. We recommend that you refer to *TME 10 Cookbook for AIX Systems Management and Networking Applications*, SG24-4867 for detailed coverage of framework installation.

Installing with SIS

Tivoli Framework Version 3.2 ships with a feature called the Software Installation Service (SIS).

SIS allows you to install TME software on managed nodes across the TMR.

We tried to use SIS to install Distributed Monitoring and the ARM Agents, but this was not possible at the time of writing, and so our installations were performed from the Tivoli Desktop.

2.1 Installing Distributed Monitoring

After installing the Tivoli Management Framework Version 3.2, we were ready to install Distributed Monitoring Version 3.5 from the product CD.

We clicked on **Install** from the toolbar on the Tivoli Desktop, and then selected **Install Product**, as you can see in Figure 4 on page 8.

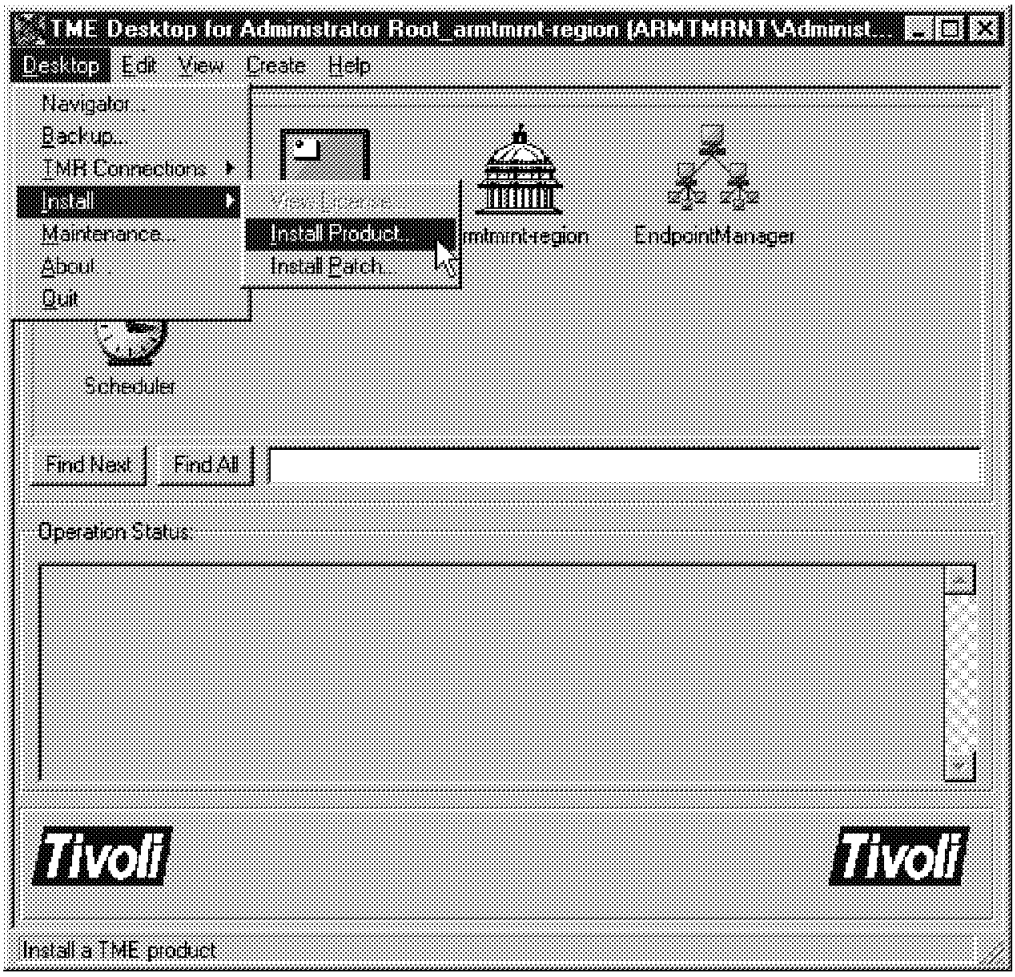


Figure 4. Starting the Distributed Monitoring Installation

You will probably need to select the correct path, and then click on **Set Media & Close**, as in Figure 5 on page 9.

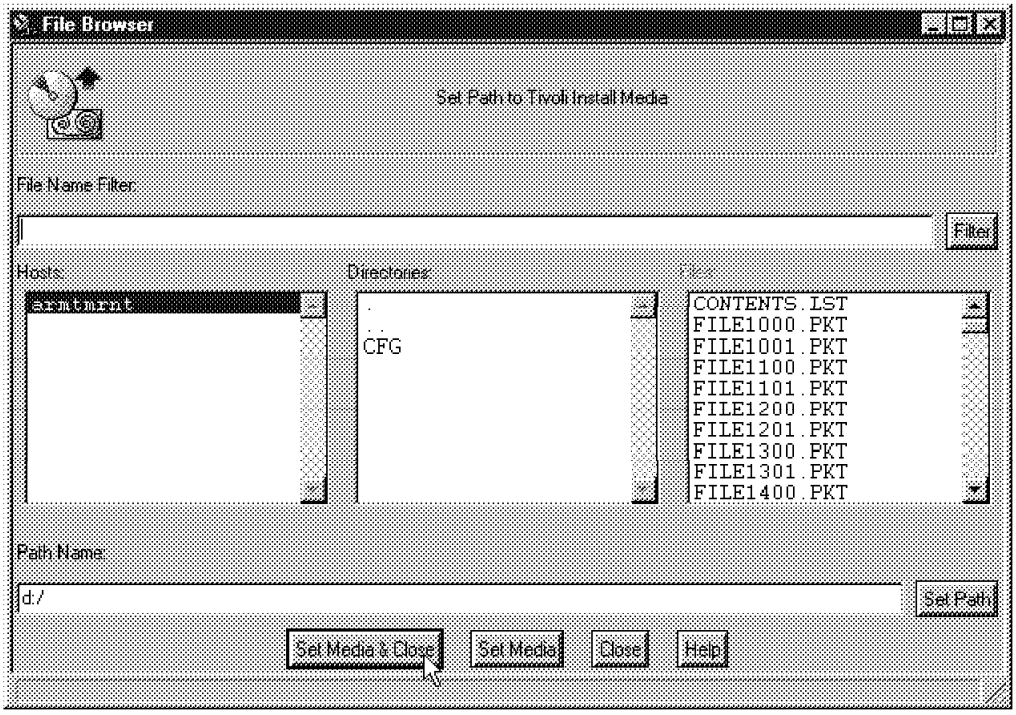


Figure 5. Setting the Installation Media

Now you will be asked to select the product that you want to install. We selected Distributed Monitoring 3.5, as you can see in Figure 6 on page 10.

Figure 6. Selecting to Install Distributed Monitoring

Now click on **Install & Close**, and you should see messages as shown in Figure 7 on page 11.

```
Checking product dependencies...
 product LCF_B is already installed as needed.
 product TMP_3.1 is already installed as needed.
Dependency check completed.
Inspecting node armtmrnt...
Installing product: TME 10 Distributed Monitoring 3.5

Unless you cancel, the following operations will be executed:
 For the machines in the independent class:
  hosts(armtmrnt)
 need to copy the machine independent generic HTML/Java files to:
    C:/Tivoli/bin/generic
 need to copy the machine independent Message catalogs to:
    C:/Tivoli/msg_cat

 For the machines in the w32-ix86 class:
  hosts(armtmrnt)
 need to copy the architecture specific Binaries to:
    C:/Tivoli/bin/w32-ix86
 need to copy the architecture specific Server Database to:
    C:/Tivoli/db/armtmrnt.db
 need to copy the architecture specific Libraries to:
    C:/Tivoli/lib/w32-ix86


---
To continue with installation, select the "Continue Install" button
 -OR-
to abort this installation attempt, select the "Cancel" button.
```

Figure 7. The Initial Installation Messages

Click on **Continue Install**. The installation will likely take a few minutes.

A successful installation will produce messages similar to those in Figure 8 on page 12.

```
Creating product installation description object...created.
Executing queued operation(s)
Distributing architecture specific Binaries for armtmrnt
 . completed.

Distributing machine independent generic HTML/Java files for armtmrnt
 . completed.


Distributing architecture specific Server Database for armtmrnt
.....................................Product install completed
successfully.
 completed.

Distributing machine independent Message Catalogs for armtmrnt
 completed.

Distributing architecture specific Libraries for armtmrnt
 completed.

Registering product installation attributes...registered.

Finished product installation
```

Figure 8. Distributed Monitoring Successfully Installed

After you install Distributed Monitoring 3.5, you may choose to continue with the installation of the Distributed Monitoring monitors; however, you do not have to install any Distributed Monitoring monitors as prerequisites for ARM monitors.

2.2 Installing ARM Agents and Monitors on the TMR Server

We inserted the CD labelled Distributed Monitoring ARM Agents.

Note: Since we were using early code, the ARM agents were shipped on a separate CD.

From the Desktop pull-down, we chose **Install**, then **Install Product**.

We had to select the correct path, and then click on **Set Media & Close**.

As you can see in Figure 9 on page 13, there are three install options:

- The TME 10 Distributed Monitoring ARM Agent option contains both the ARM Server Agent and ARM Client Agent.
- The TME 10 Distributed Monitoring ARM Monitors option is the ARM monitoring collection.
- The TME 10 Distributed Monitoring ARM Endpoint consists of files that are necesary to install the ARM Client Agent on LCF endpoints.

Figure 9. The ARM Agent Install Options

We installed the ARM Agent and ARM Monitors options. We did not yet need the ARM Endpoint option, because at this point we were installing everything on one machine, the TMR server.

Note: The options must be installed one at a time; it is not possible to select more than one install option in the dialog box.

When you choose the **ARM Agent** option, you will see a pop-up as shown in Figure 10 on page 14.

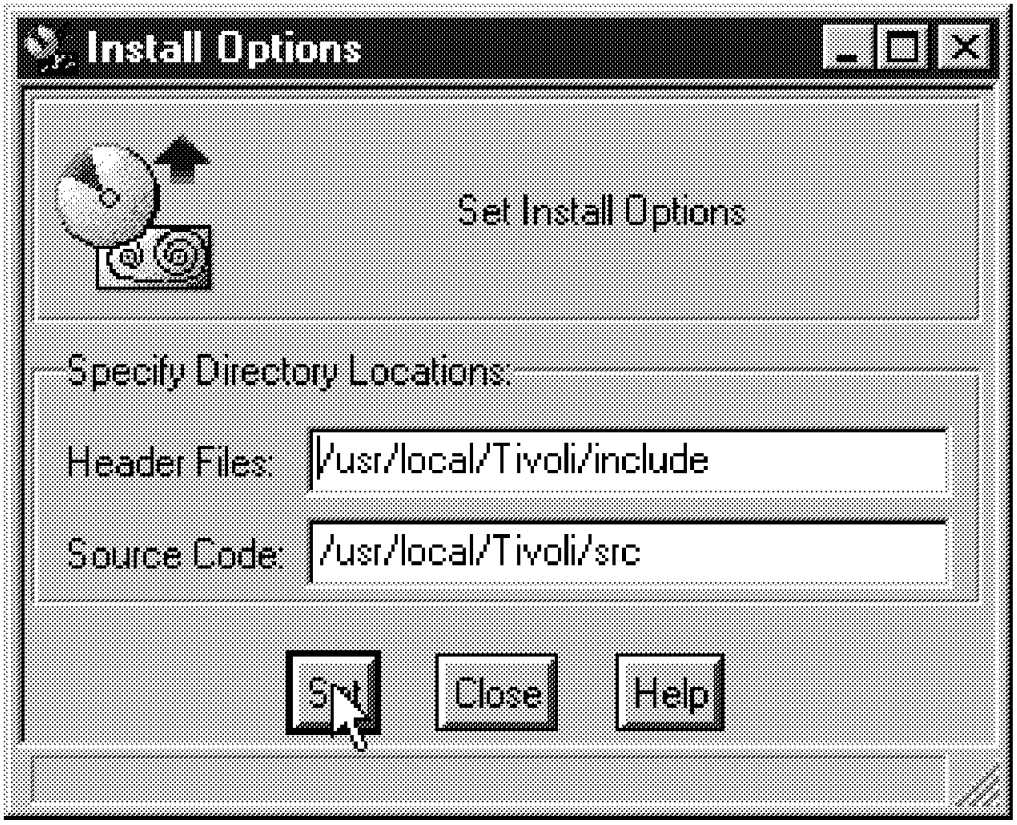


Figure 10. Selecting the Locations for the Source and Header Files

Forward slashes

Don't be perturbed by the forward slashes in the directory paths shown in this dialog box. The same dialog box is used whether you are installing on UNIX or Windows, and the paths are coded for a UNIX environment. Even though forward slashes are not allowed in Windows NT, the install completes successfully.

When you click on **Set**, the following files are installed:

Path and File Name	File Size
usrlocalTivoliincludew32-ix86tivoliarm.h	7 KB
usrlocalTivoliincludew32-ix86tivoliarm16.h	3 KB
usrlocalTivolisrcarmsample.c	24 KB
usrlocalTivolisrcarmsample.mak	0.4 KB

The header files, with the .h extension, can be used to compile ARM-instrumented applications for Windows; see 8.2, "Compiling with ARM" on page 106 and 8.3, "Compiling 16-bit Applications for Windows 95 and NT" on page 108, for details on how to do this. They are not needed when you run an ARM-instrumented application.

The sample.c file is C language source code for an example application that is ARM-instrumented.

The sample.mak file is a make file that can be used to compile the sample.c source code on UNIX platforms.

The files are listed in full in Appendix A, “Source File Listings” on page 115.

Reading the files

We found that these files cannot easily be viewed with the Notepad in Windows 95/NT. They need to be viewed using Wordpad. This should be fixed by the time you read this book.

We checked the installation messages, to ensure that there were no errors. You can see the messages we got for the ARM Agent installation in Figure 11.

```
     C:/Tivoli/bin/w32-ix86
  need to copy the architecture specific Header Files to
     armtmrnt/usr/local/Tivoli/include/w32-ix86


---

To continue with installation, select the "Continue Install" button
 -OR-
to abort this installation attempt, select the "Cancel" button.

Creating product installation description object...created.
Executing queued operation(s)
Distributing architecture specific Libraries for armtmrnt
 completed.

Distributing architecture specific Binaries for armtmrnt
 completed.

Distributing architecture specific Header Files for armtmrnt
 completed.

Distributing machine independent Message Catalogs for armtmrnt
 completed.

Distributing machine independent Source Code for armtmrnt
 completed.

Registering product installation attributes...registered.

Finished product installation.
```

Figure 11. Messages from the ARM Agent Installation

After installing the ARM Monitors option, we recycled the TMR server by closing and then restarting the Tivoli Desktop. Now we could see a new policy region icon on the TMR Desktop, as shown in Figure 12 on page 16.

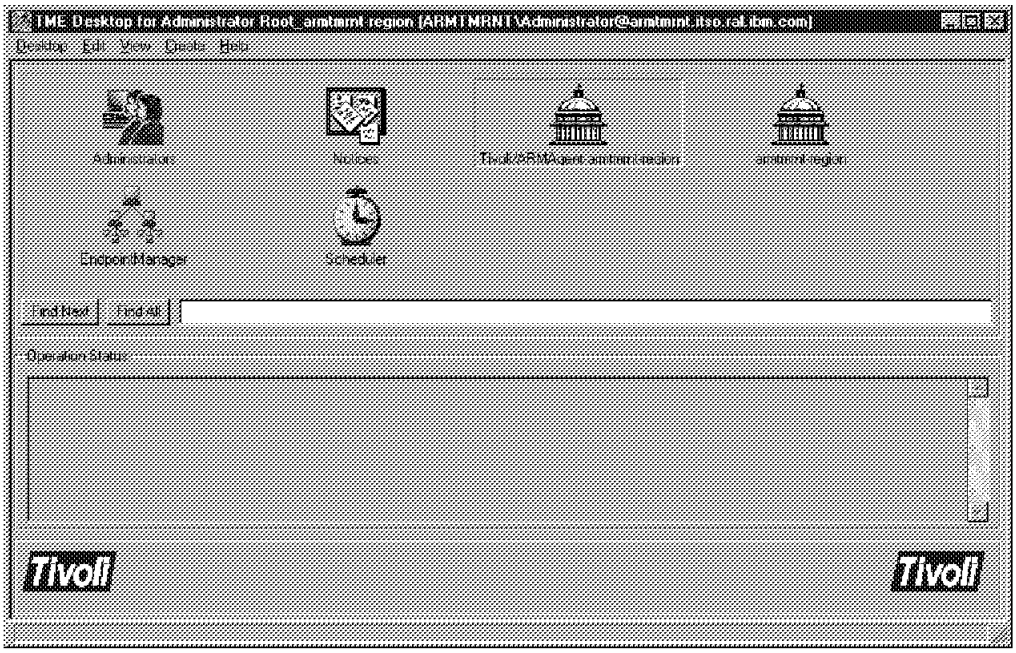


Figure 12. The ARM Policy Region on the TME Desktop

We had a policy region named `Tivoli/ARMAgent-armtmrnt-region`. The region contains the ARM Library that will be used to manipulate ARM tasks, as you can see in Figure 13.

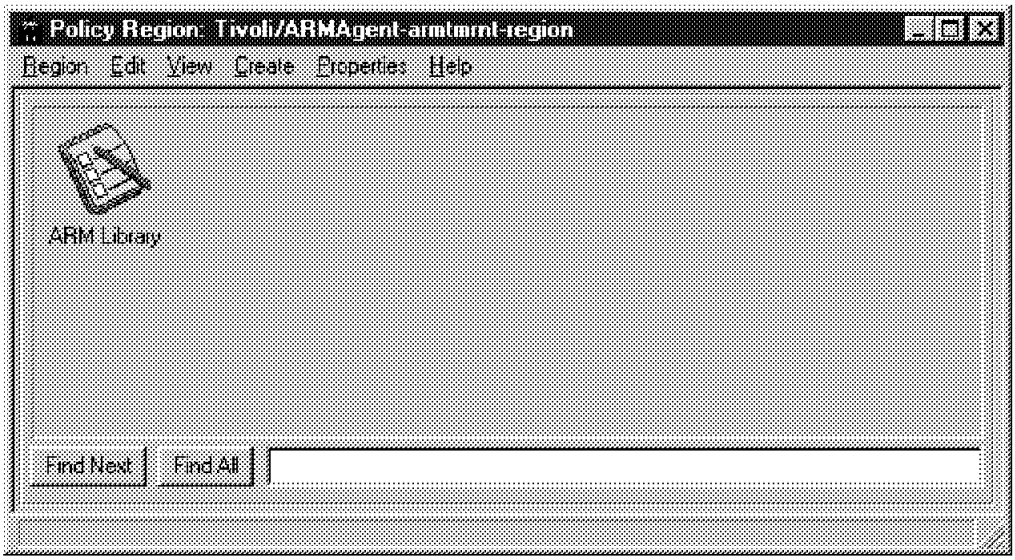


Figure 13. The New ARM Library

When you open the ARM Library icon you will see a variety of tasks, as shown in Figure 14 on page 17.

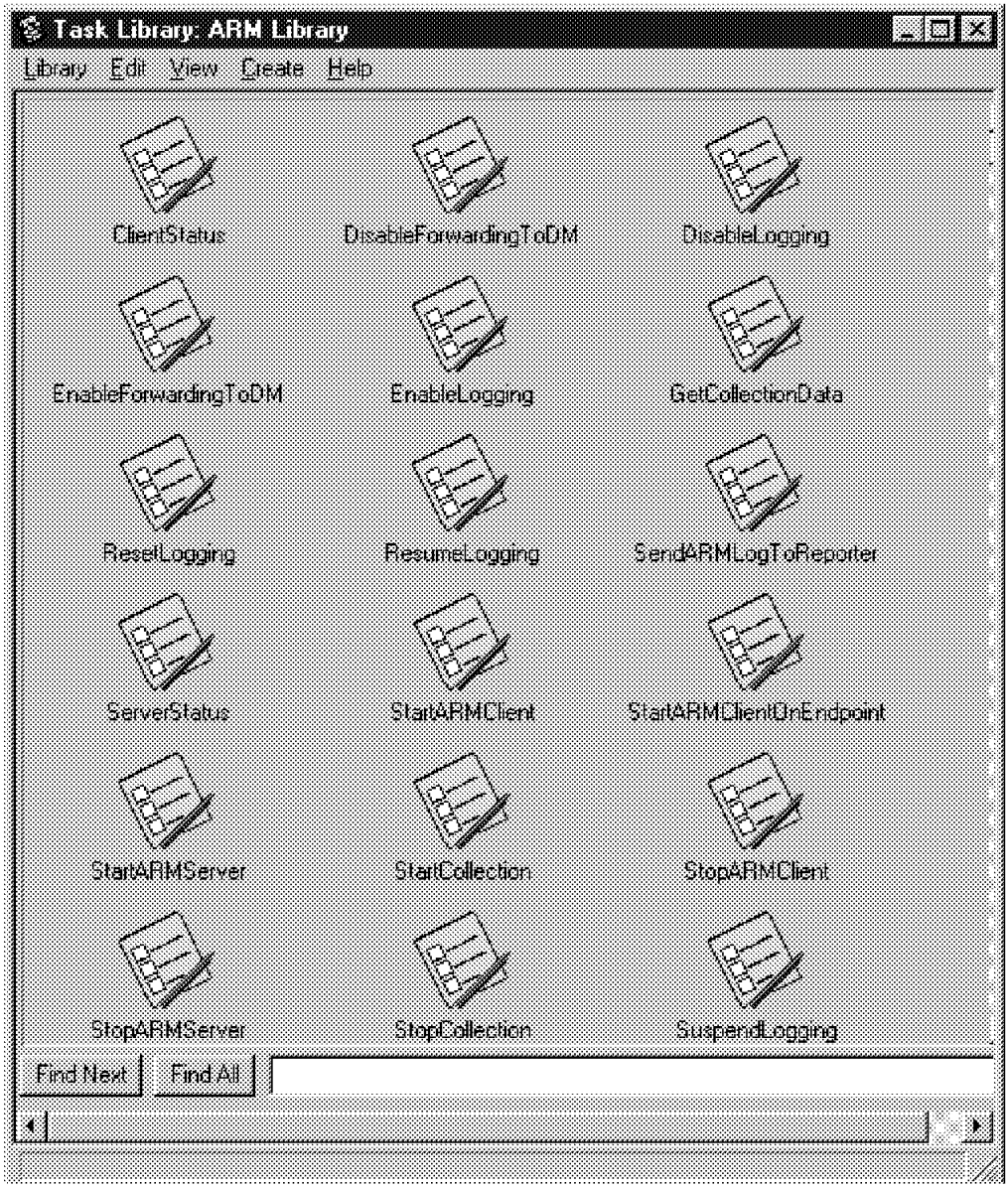


Figure 14. The Contents of the ARM Library

It is from this screen that you can execute ARM tasks.

Each task is a script or batch file that resides in the directory `\tivoli\bin\xxxx\Tas\TASK_LIBRARY\bin\yyyy\`, where xxxx is the operating system name and yyyy is a combination of numbers and letters.

2.3 Creating A Distributed Monitoring Proxy

In order to use the graphical monitoring or event generation capabilities of Distributed Monitoring, you will need to create monitors for the ARM Client Agent. This requires that you create a Distributed Monitoring proxy for the client, and use the special ARM environment variable, ARMCLIENT, to associate it with the ARM Server Agent machine where the Sentry engine runs.

You can find a fuller discussion of this in 3.1, “Before You Start - the Sentry Proxy” on page 24.

To create a proxy, open the ARM policy region from the Tivoli Desktop, then use the Create pull-down to create a SentryProxy, as in Figure 15.

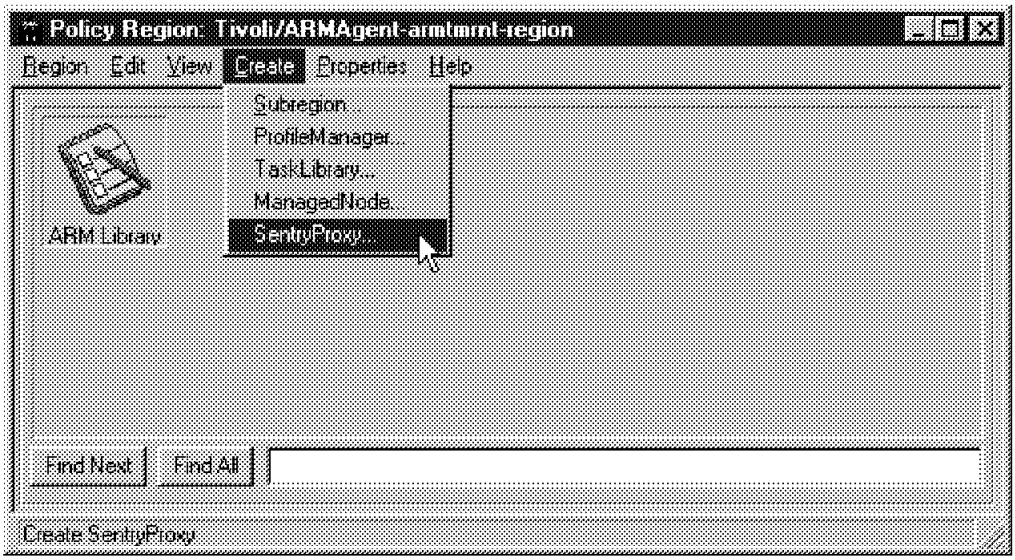


Figure 15. Creating a Distributed Monitoring Proxy Endpoint

Distributed Monitoring versus Sentry

This naming standard is a little confusing because it follows the old naming standard of Sentry. You are actually creating a Distributed Monitoring proxy endpoint.

Give the proxy endpoint a unique name - we recommend that you do not use the machine′s IP hostname - then choose a managed node that this proxy endpoint will reside on.

In our example, shown in Figure 16 on page 19, we called the Distributed Monitoring proxy endpoint tmr_proxy, and it resides on the managed node armtmrnt, which is our TMR server.

Figure 16. Creating the Proxy Endpoint

Now click on **Create & Close**.

Figure 17 on page 20 shows the icon created for a proxy endpoint.

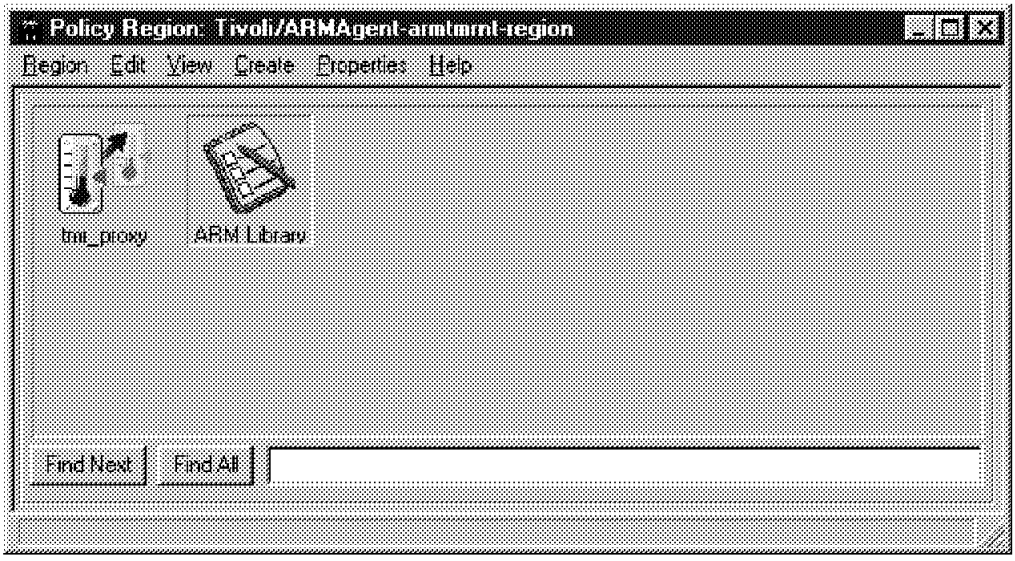


Figure 17. The New Proxy Endpoint in Our Policy Region

It is now necessary to configure the proxy endpoint's environment to make the special ARM variable, ARMCLIENT, available.

Open the newly created Proxy Endpoint icon, then click on the **Configure** option on the toolbar, and select **Set Environment**.

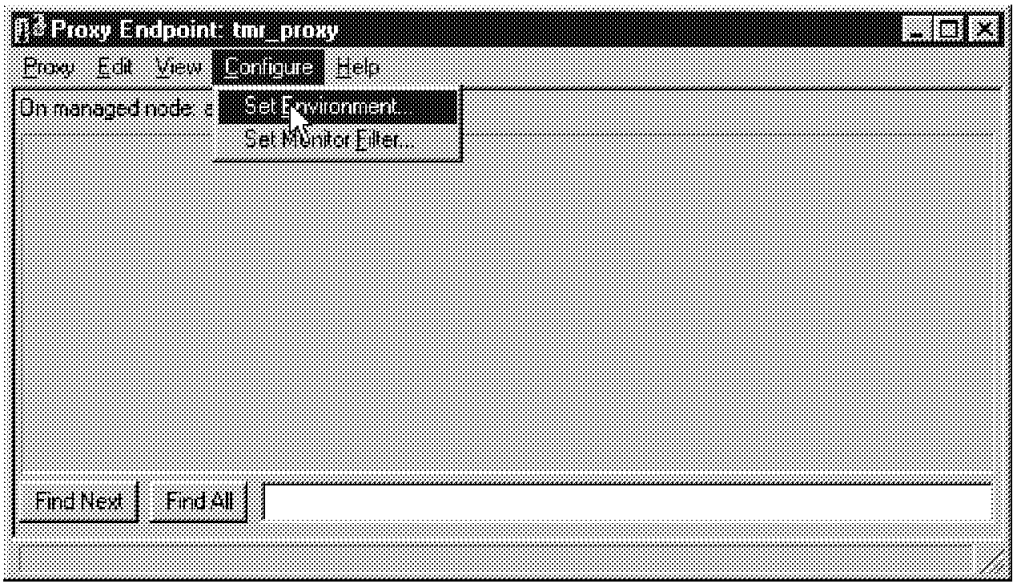


Figure 18. Configuring the Proxy Endpoint

Now you can edit the set of environment variables that belongs to this Proxy Endpoint. Specify ARMCLIENT as the Name of the environment variable, and specify the hostname of the machine where the ARM client agent will run as the Value.

Now click on **Add/Set**, so that the ARMCLIENT variable is set. The screen should look similar to Figure 19 on page 21.

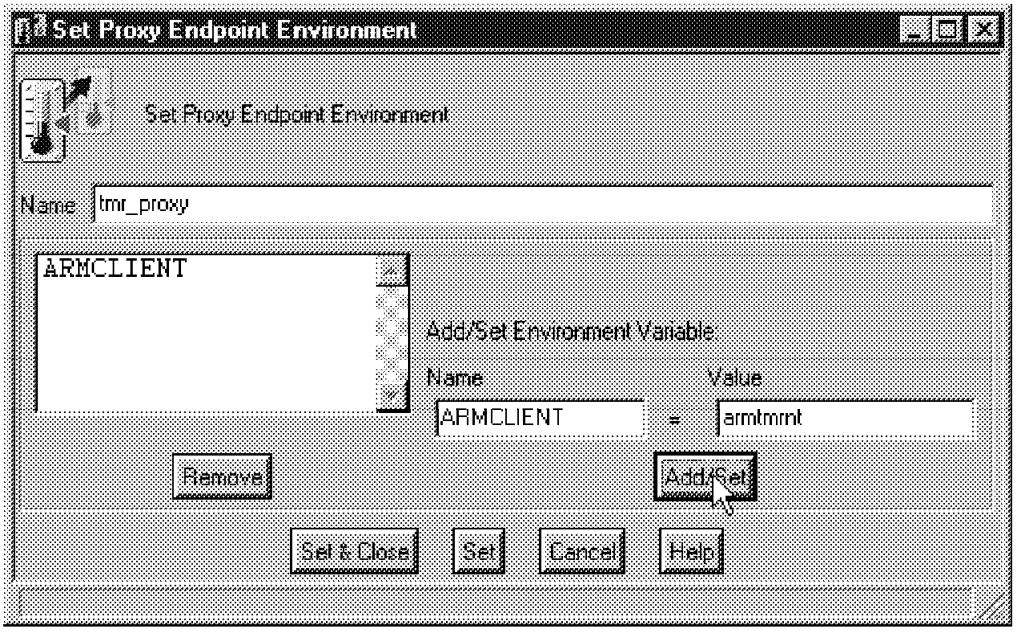


Figure 19. Setting the Proxy Endpoint Environment

Now click on the **Set & Close** button, to return to the Proxy Endpoint: tmr_proxy dialog box. Then, from the Proxy pull-down menu, select **Close** to return to the Policy Region window.

Chapter 3. Installing ARM Agents on ″Managed″ Machines

The ARM Agents are available for a number of different operating systems as you can see in Figure 20.

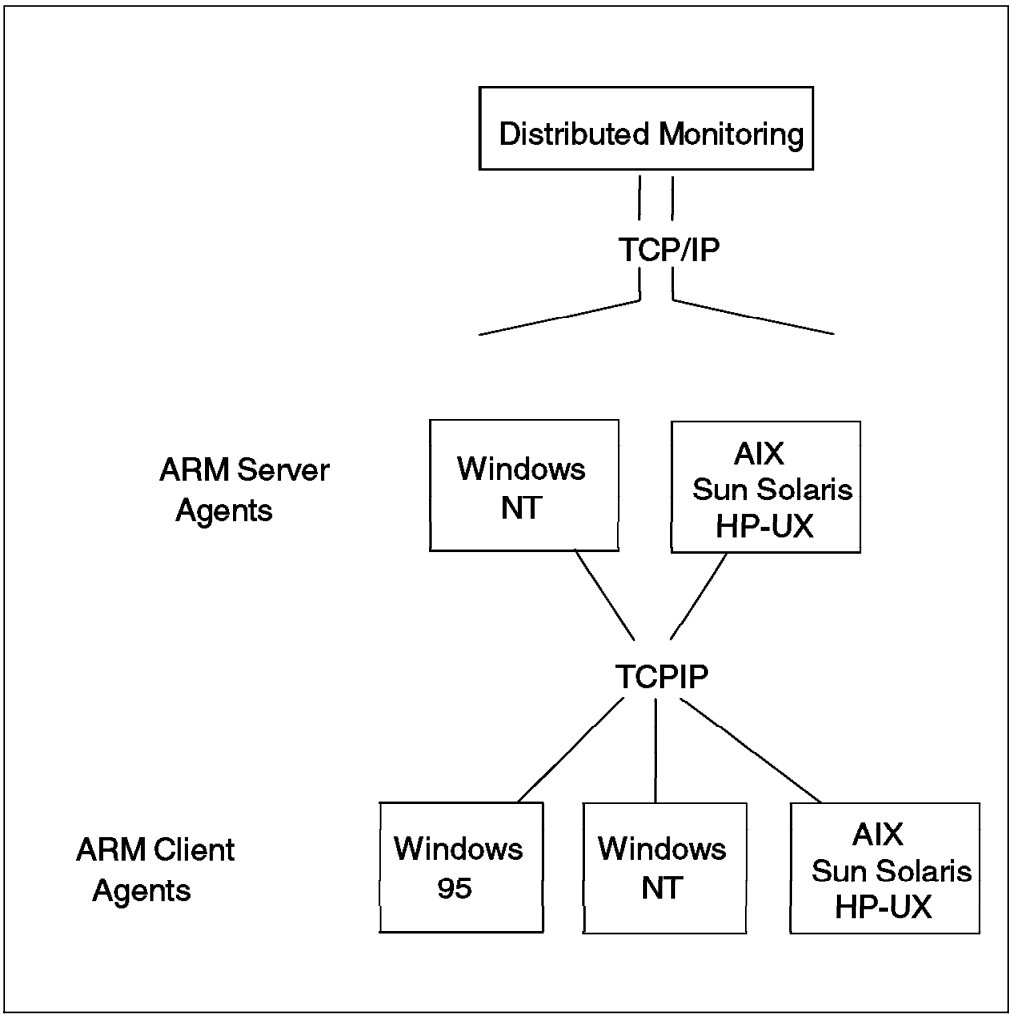


Figure 20. ARM Server to ARM Client Communications

In 2.2, “Installing ARM Agents and Monitors on the TMR Server” on page 12 we installed both the ARM Server Agent and the ARM Client Agent on one machine. You *must* install the ARM Client Agent on each machine where you want to measure response times, but you do not need to install an ARM Server Agent on each of these machines.

Many ARM Client Agents can report to one ARM Server Agent, so it makes sense to install and set up the ARM Server Agent only where you need it.

3.1 Before You Start - the Sentry Proxy

In order to perform the following functions, you need to be able to distribute ARM monitors to the Sentry engine:

- Graphical monitoring
- Asynchronous monitoring

The Sentry engine must be running on a machine where the ARM Server Agent is installed.

Therefore, if on any machine you run only the ARM Client Agent, you will need to perform the following steps:

1. Create a Sentry Proxy on a managed node where the ARM Server Agent is running.
2. Create the ARMCLIENT environment variable on that Sentry Proxy, to identify the ARM Client Agent machine to the ARM Server Agent.

This configuration is shown in Figure 21 and the procedure in 2.3, “Creating A Distributed Monitoring Proxy” on page 17 shows how to set it up.

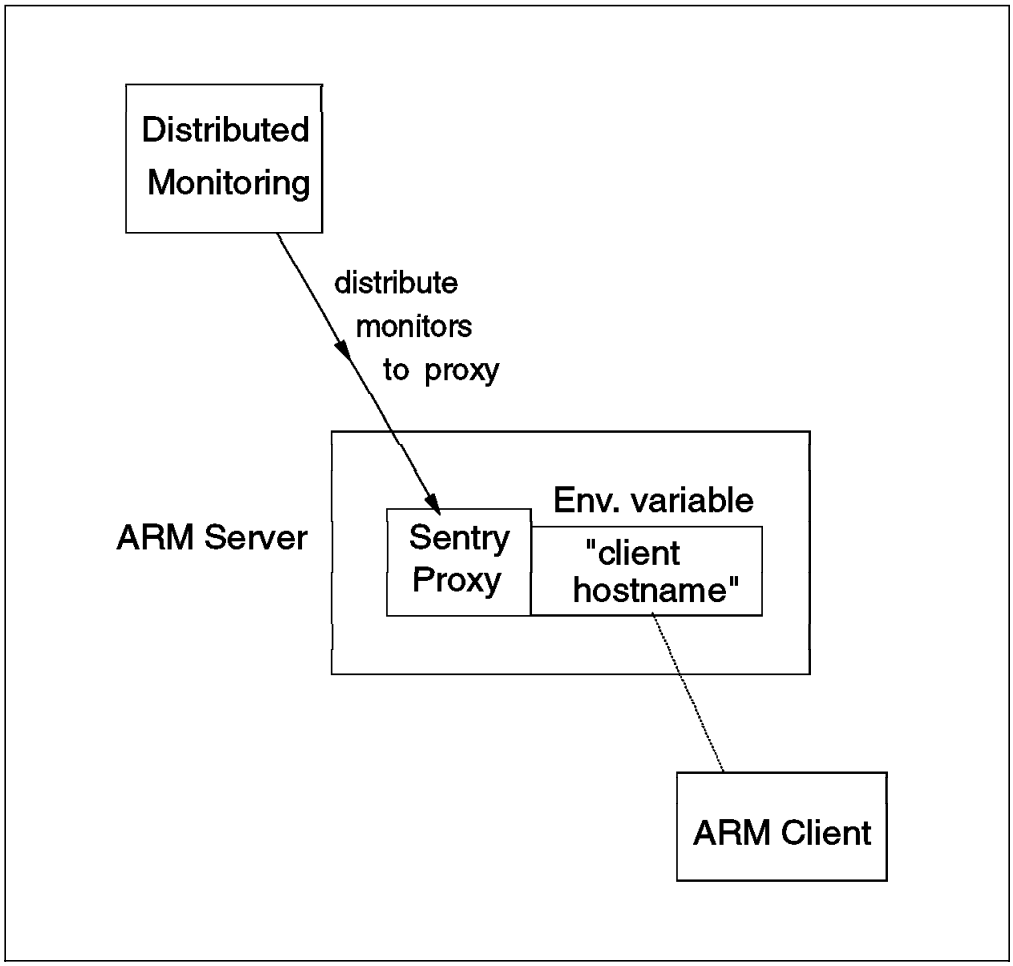


Figure 21. Monitoring an ARM Client with a Sentry Proxy

This will allow monitors to run against the ARM Client Agent machine.

Unfortunately this means that if you want to connect the ARM Client Agent to a different ARM Server Agent - maybe because the original ARM Server Agent machine has failed - you must perform the following steps before restarting the ARM Client Agent with the -h flag set to point to the other ARM Server Agent machine.

1. Create a Sentry Proxy on the other ARM Server Agent machine, to represent the ARM Client Agent machine.
2. Create the ARMCLIENT environment variable on that Sentry Proxy, to identify the ARM Client Agent machine to the ARM Server Agent.
3. Distribute the ARM monitors to the new Sentry Proxy.

3.2 Installing the ARM Client Agent on a Windows NT Managed Node

Before you can install the ARM Agents, you must install Distributed Monitoring Version 3.5 on the managed node, just as in 2.1, “Installing Distributed Monitoring” on page 7.

We inserted the CD labelled Distributed Monitoring ARM Agents in the CD drive on our TMR server machine, and from the Desktop pull-down, we clicked on **Install**, then **Install Product**.

We selected the install option **TME 10 Distributed Monitoring ARM Agent**, as you can see in Figure 22 on page 26.

Figure 22. Selecting to Install the Agents

Both Agents must be installed

The ARM Agent option will install both the ARM Server Agent and the ARM Client Agent. It is not possible to install only one of them.

Now you will see a pop-up similar to Figure 23 on page 27.

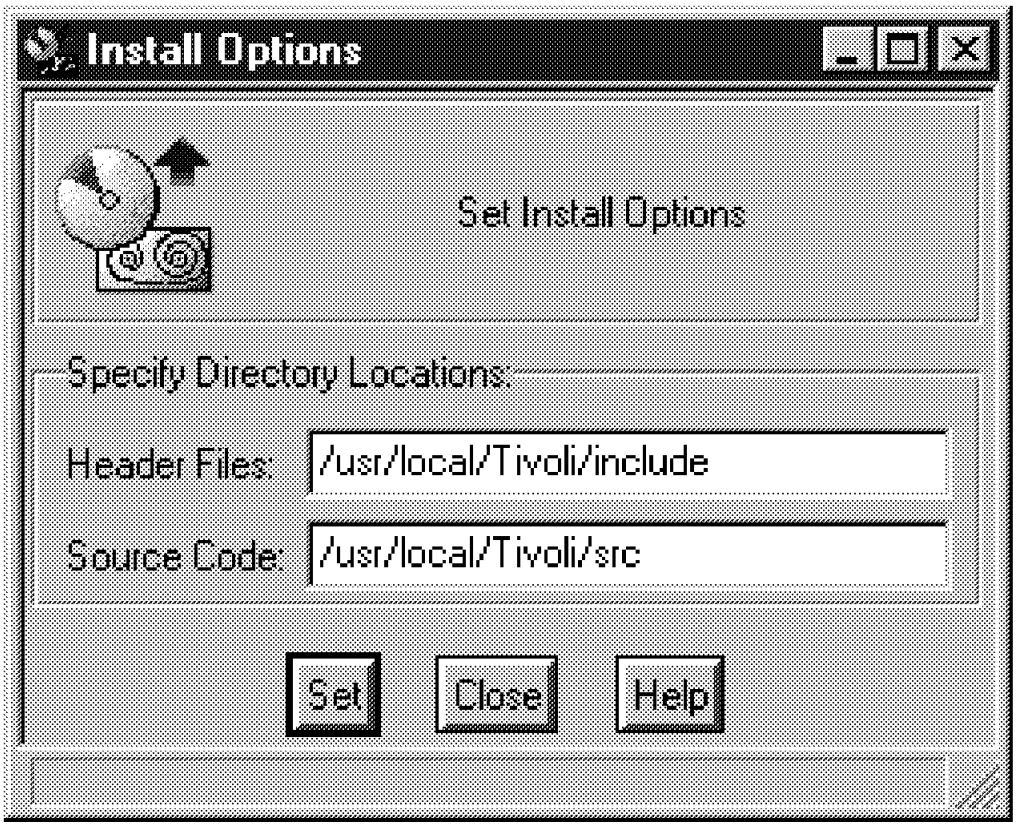


Figure 23. Selecting the Locations for the Source and Header Files

When you click on **Set**, the following files are installed:

```
usrlocalTivoliincludew32-ix86tivoliarm.h
\usr\local\Tivoli\include\w32-ix86\tivoli\arm16.h
\usr\local\Tivoli\src\arm\sample.c
\usr\local\Tivoli\src\arm\sample.mak
```

Now click on **Install & Close**. Then, after checking the messages that appear, click on **Continue Install**.

The ARM library, libarm32.lib will now be installed in the `usrlocalTivolilibw32-ix86` directory.

3.3 Installing the ARM Client Agent on a PC Managed Node

We used the ARM Agents CD to install the ARM Client Agent on two PC Managed Nodes; one was running Windows NT Workstation, and the other was running Windows 95.

All the files that are required for this installation are found in the PC directory on the ARM Agents CD. For installing the ARM Client on PCs in a production environment, you may want to copy the contents of this directory onto a diskette, and use the diskette for the installations, or use Tivoli Courier to install the Agent remotely.

There were 11 files in the PC directory on our ARM Agents CD. This included two hidden files; if you do copy the PC directory onto a diskette, make sure that these files are copied.

We ran the Setup program from the PC directory on the ARM Agents CD.

After the informational welcome screen, you will be asked to confirm the installation directory. We allowed this to default, as you can see in Figure 24.

Figure 24. Choosing the Installation Directory

The PATH statement will then be updated, as you can see in Figure 25.

Figure 25. The PATH Statement Updated

Click on **OK**, and the NLSPATH environment variable will be updated, as you can see in Figure 26 on page 29.

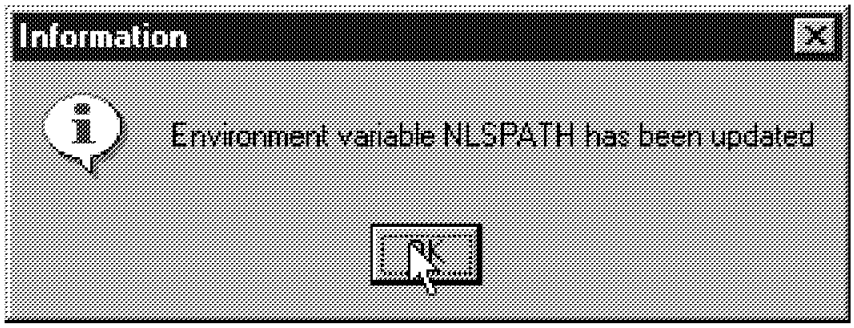


Figure 26. The NLSPATH Updated

Now click on **OK**, so that the installation completes. You will then be required to reboot the PC.

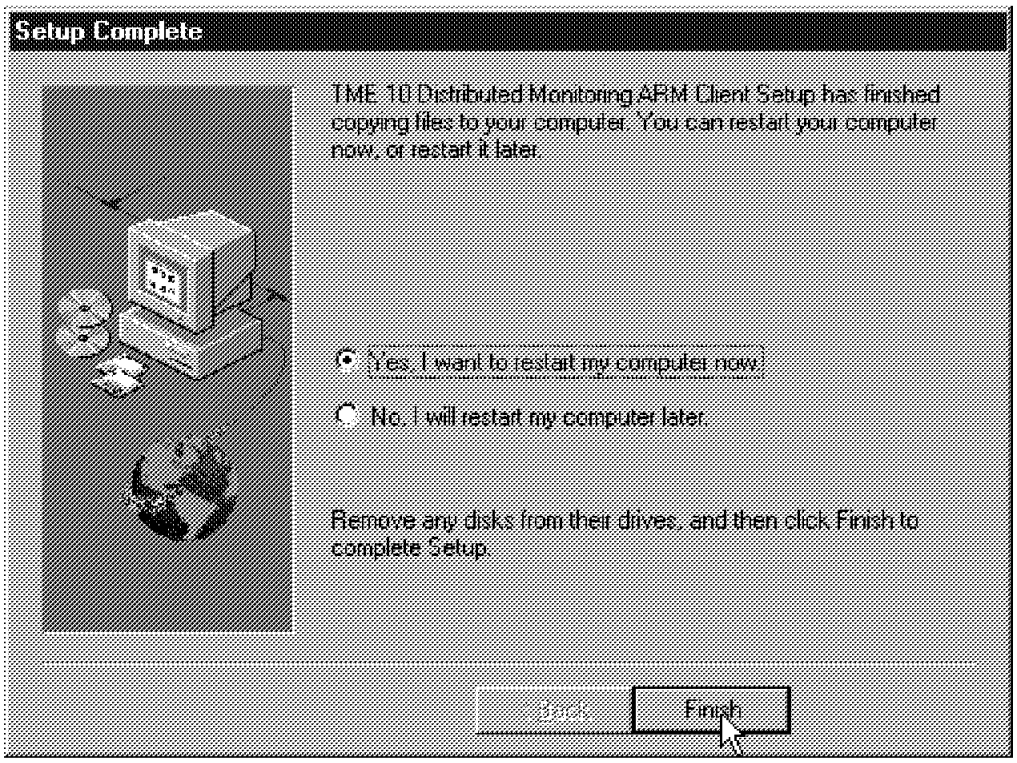


Figure 27. The Installation Completed

Remember to remove the diskette from the drive before rebooting!

We found that, on Windows NT Workstation, the PATH was updated without a ; delimiter, as you can see.

```
PATH=C:\WINNT\system32;C:\WINNT;C:\Program Files\Personal CommunicationsC:\Tivoli\arm\bin;
```

We added the ; delimiter by editing the PATH in the Environment tab of the System folder in the Control Panel. This allowed warmcmd commands to be issued from any command prompt.

The following runtime library files, required when you run an ARM-instrumented application, were added by the installation program:

Tivoliarmliblibarm32.dll
\Tivoli\arm\lib\libarm16.dll

The Tivoliarmlib directory is not normally in the Windows libpath, and so when an ARM-instrumented application runs, its ARM API calls will fail. We recommend that you copy the DLL files to the appropriate directory as follows, so that they will always be found:

Table 1. The Correct Directory Locations for the Runtime DLLs

DLL name	Windows 95	Windows NT
libarm32.dll	WindowsSystem	WINNTSystem32
libarm16.dll	WindowsSystem	WINNTSystem

The following entries were added to the Windows 95 AUTOEXEC.BAT file:

```
SET PATH=C:\TIVOLI\ARM\BIN
SET NLSPATH=C:\TIVOLI\ARM\MSG_CAT\%%L\%%N.CAT
SET ARMNLSPATH=C:\TIVOLI\ARM\MSG_CAT\%%L\%%N.CAT
```

The header and library files that are required for compiling ARM-instrumented applications are *not* installed by the installation program for PC Managed Nodes. If you want to compile ARM-instrumented applications on such a machine, you will need to get the files from a managed node on which the ARM Agents have been installed.

3.4 Installing the ARM Server and Client Agents on a Sun Managed Node

The ARM Agents can run on Solaris 2.4, or later. Check the level of the operating system on your Sun machine by issuing the command `uname -a`.

Before you can install the ARM Agents, you must install Distributed Monitoring Version 3.5 on the Sun Managed Node, just as in 2.1, “Installing Distributed Monitoring” on page 7.

We inserted the CD labelled Distributed Monitoring ARM Agents in the CD drive on our TMR server machine, and from the Desktop pull-down, we clicked on **Install**, then **Install Product**.

We selected the install option **TME 10 Distributed Monitoring ARM Agent**, as you can see in Figure 28 on page 31.

Figure 28. Selecting to Install the Agents

Both Agents must be installed

The ARM Agent option will install both the ARM Server Agent and the ARM Client Agent. It is not possible to install only one of them.

Now you will see a pop-up similar to Figure 29 on page 32.

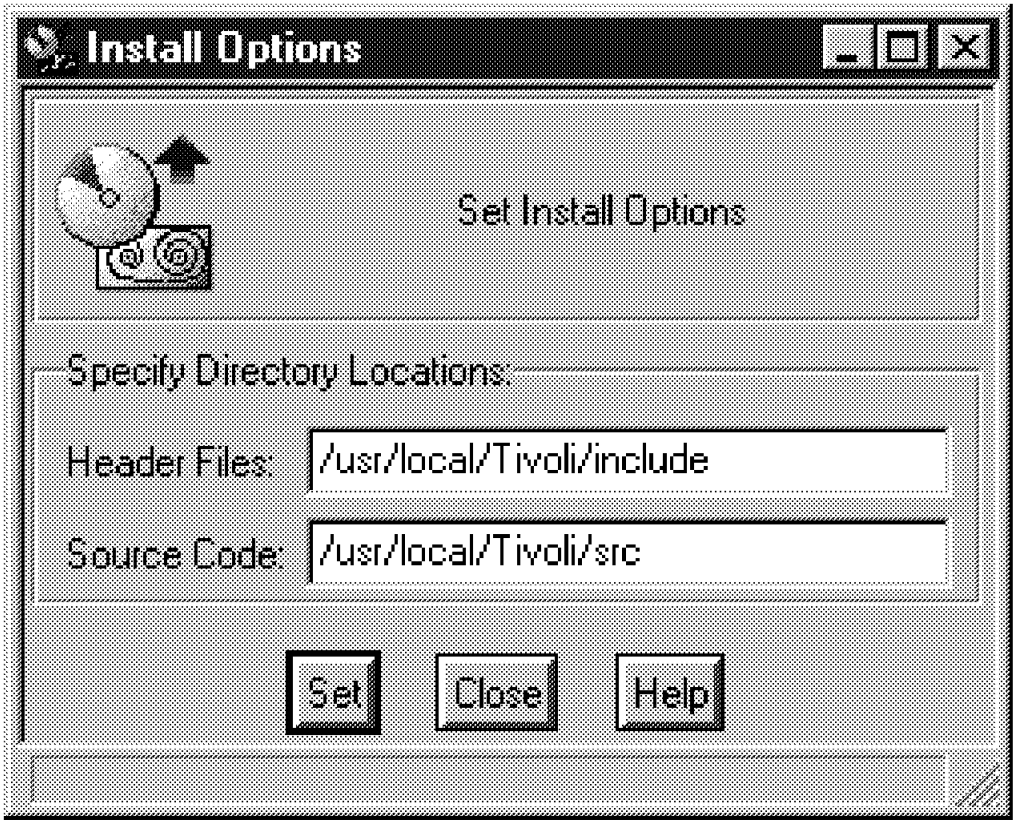


Figure 29. Selecting the Locations for the Source and Header Files

When you click on **Set**, the following files are installed:

```
/usr/local/Tivoli/include/solaris2/tivoli/arm.h
/usr/local/Tivoli/src/arm/sample.c
/usr/local/Tivoli/src/arm/sample.mak
```

Now click on **Install & Close**. Then, after checking the messages that appear, click on **Continue Install**.

The ARM library, libarm.so, will now be installed in the `/export/home/Tivoli/lib/solaris2` directory.

3.5 Installing the ARM Server and Client Agents on an AIX Managed Node

The ARM Agents can run on AIX Version 3.2.5, or later.

Before you can install the ARM Agents, you must install Distributed Monitoring Version 3.5 on the AIX Managed Node, just as in 2.1, “Installing Distributed Monitoring” on page 7.

We inserted the CD labelled Distributed Monitoring ARM Agents in the CD drive on our TMR server machine, and from the Desktop pull-down, we clicked on **Install**, then **Install Product**.

We selected the install option **TME 10 Distributed Monitoring ARM Agent**.

Both Agents must be installed

The ARM Agent option will install both the ARM Server Agent and the ARM Client Agent. It is not possible to install only one of them.

The installation copied the following header and library files to our AIX 4.2 machine.

```
/usr/local/Tivoli/src/include/aix4-r1/tivoli/arm.h
```

```
/usr/local/Tivoli/lib/aix4-r1/libarm.a
```

In order to be able to issue warmcmd commands from the AIX shell, the Tivoli environment variables need to be set. We did this by adding the following line to the .profile file of the user ID that the TMR server uses to connect with the managed node - in our case this was *root*.

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

Chapter 4. Using the ARM Agents

Now that you have the agents installed, and the basic configuration is done, you can start them up, and start up collection for ARM-instrumented applications.

4.1 Before You Start

When working with the ARM agents on Windows NT, you may see a screen similar to Figure 30.

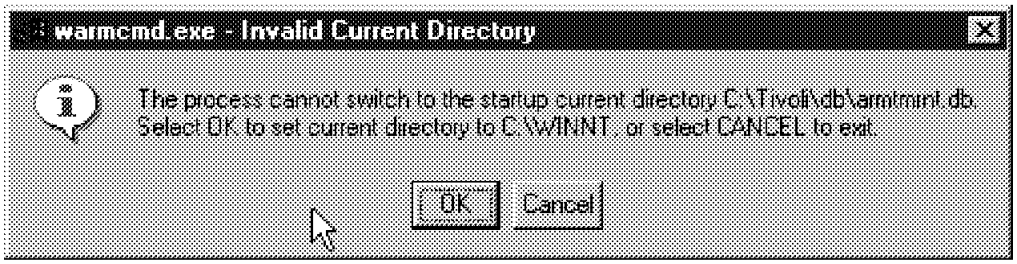


Figure 30. The Directory Permissions Error

You can click on **OK**, and continue, but if you don't want to be bothered by the message, complete the steps in Appendix F, "Granting Permissions for the Tivolidb Directory on NT" on page 137 to grant sufficient permissions on the Tivolidb directory, for the user ID tmersrvd.

If you want to control the ARM agents from the command line, this must be done from a window that has the Tivoli environment variables set. See Appendix E, "Setting the Tivoli Environment Variables for a DOS Window on NT" on page 135 for instructions on how to do this.

4.2 Starting the ARM Server Agent

The ARM Server Agent can be started from the command line, on the machine where it will run, but in a production environment it will probably be more convenient to start it remotely using a Tivoli task.

4.2.1 Starting from the Command Line

Type `warmcmd srvstart`, to start the server. This command must be issued on the machine where you want the arm Server Agent to start.

The optional arguments are:

-f logfilename Allows you to name a file where collected data can be logged, so that it can later be used as input to Reporter. See Chapter 5, "Reporting ARM Data with Reporter" on page 53 for details.

-s Forwards any asynchrous monitor events to Distributed Monitoring

Since you will not see any returned messages to indicate that the agent has started, we recommend that you immediately check on its status.

4.2.2 Checking the Server Status

Type `warmcmd srvstatus -v3`, on the ARM server machine, to check the status of the ARM Server Agent. You should see messages similar to the following:

```
The ARM Server Subagent is active
Data logging disabled
Forward asynchronous events to DM: disabled
```

If you look in the message log for the ARM Server Agent, asxrtag.log, you should see the message:

```
11/11/1997 16:04:46 - ASX0116I The ARM Server Subagent is running.
```

This is the log file for the ARM Server Agent, and the messages in the file can be very helpful in isolating any problems that you may encounter.

Note that the directory path in which this file resides will vary, depending on the platform and the Tivoli installation; rather than list them all here, we suggest that you use the Find command to locate the file.

You can also use the netstat command to check that the server is listening for ARM clients on TCP port 2600. This is the default port; if you need to change it, follow the instructions in *TME 10 Distributed Monitoring ARM Agents User's Guide*.

```
C:\>netstat -a|more

Active Connections

  Proto  Local Address          Foreign Address        State

  TCP    armtmrnt:2600          0.0.0.0:0              LISTENING
```

4.2.3 Starting from the Tivoli Desktop

To start the ARM Server Agent from the TME Desktop, open the policy region for the ARM agents.

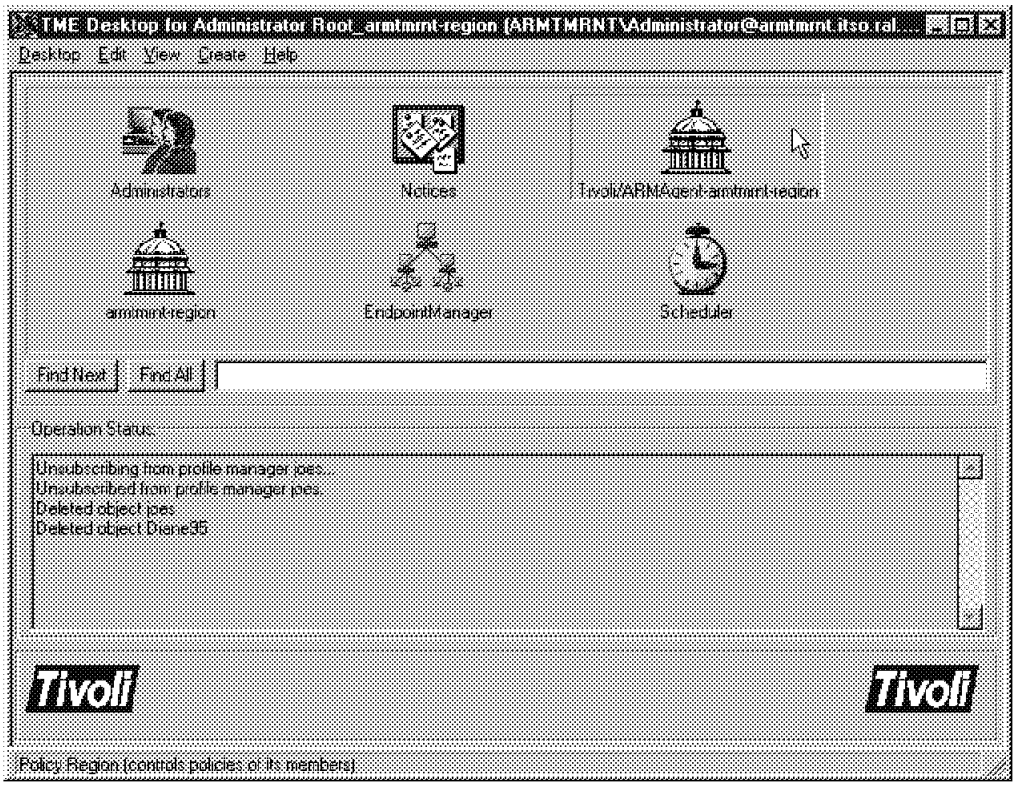


Figure 31. Opening the Policy Region for the ARM Agents

Now open the **ARM Library**.

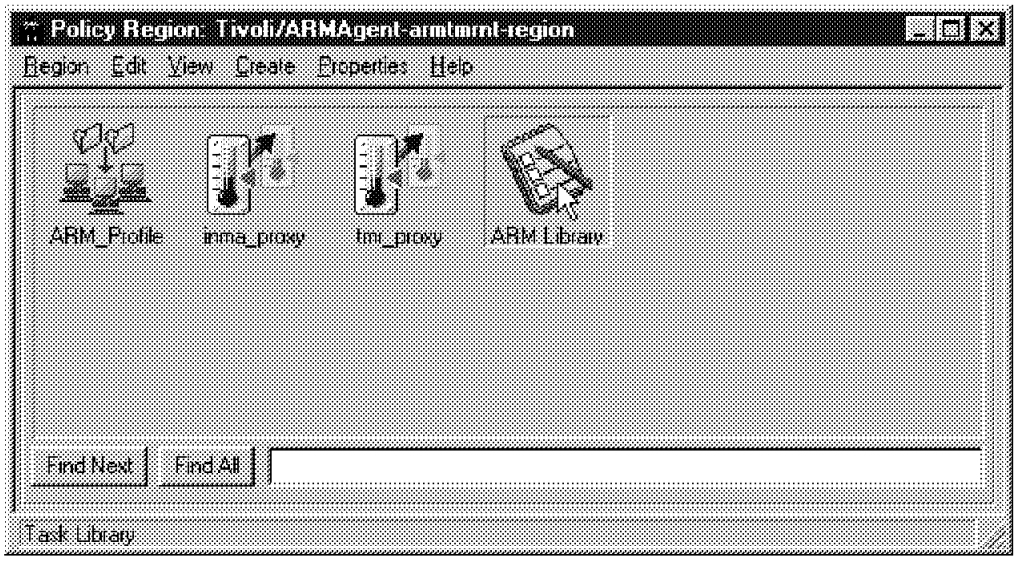


Figure 32. Opening the ARM Task Library

Open the **StartARMServer** task.

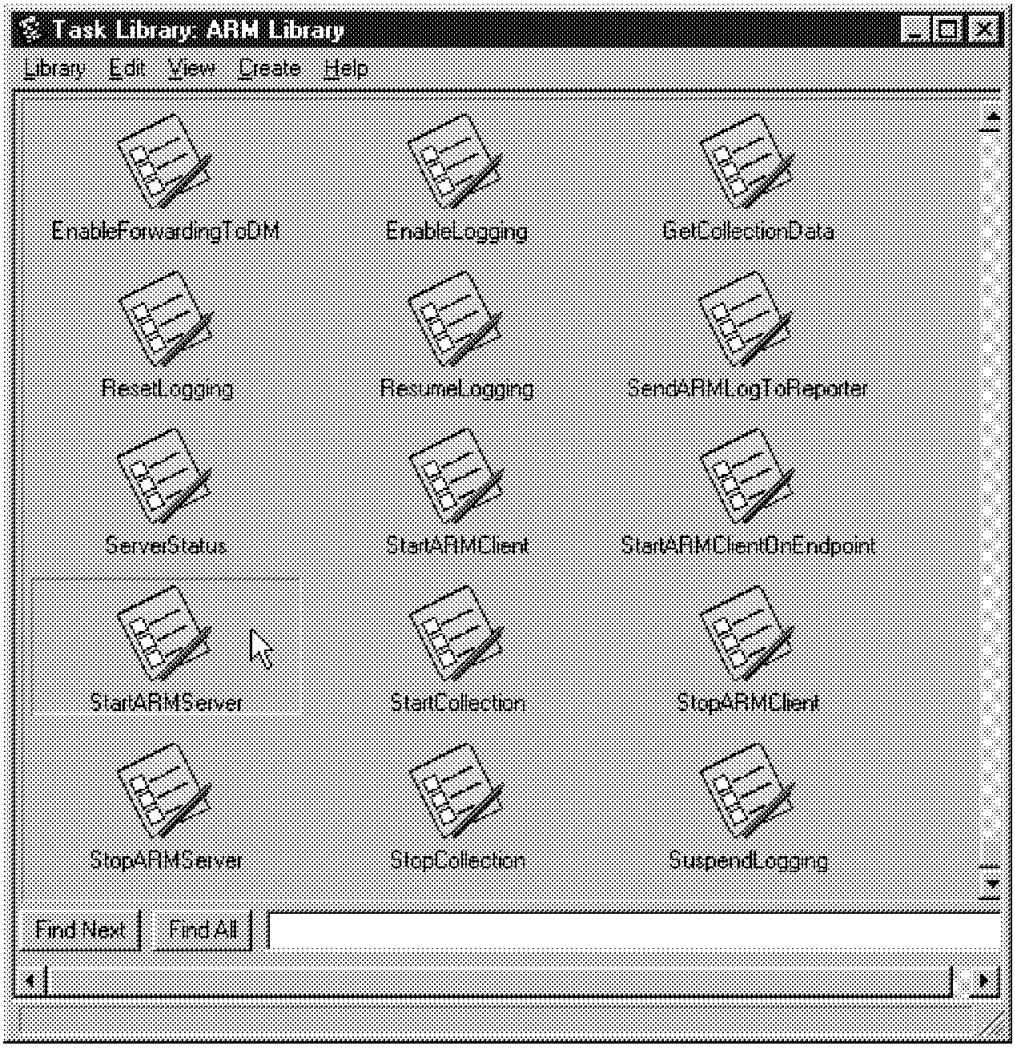


Figure 33. Opening the StartARMServer Task

Choose the machine where the task will execute, from the list of available task endpoints.

If you do not check an Output Destination from this screen, a dialog box will pop up, stating that you have not selected any destination for the output messages, and ask if you want to ignore the output from the execution of the task.

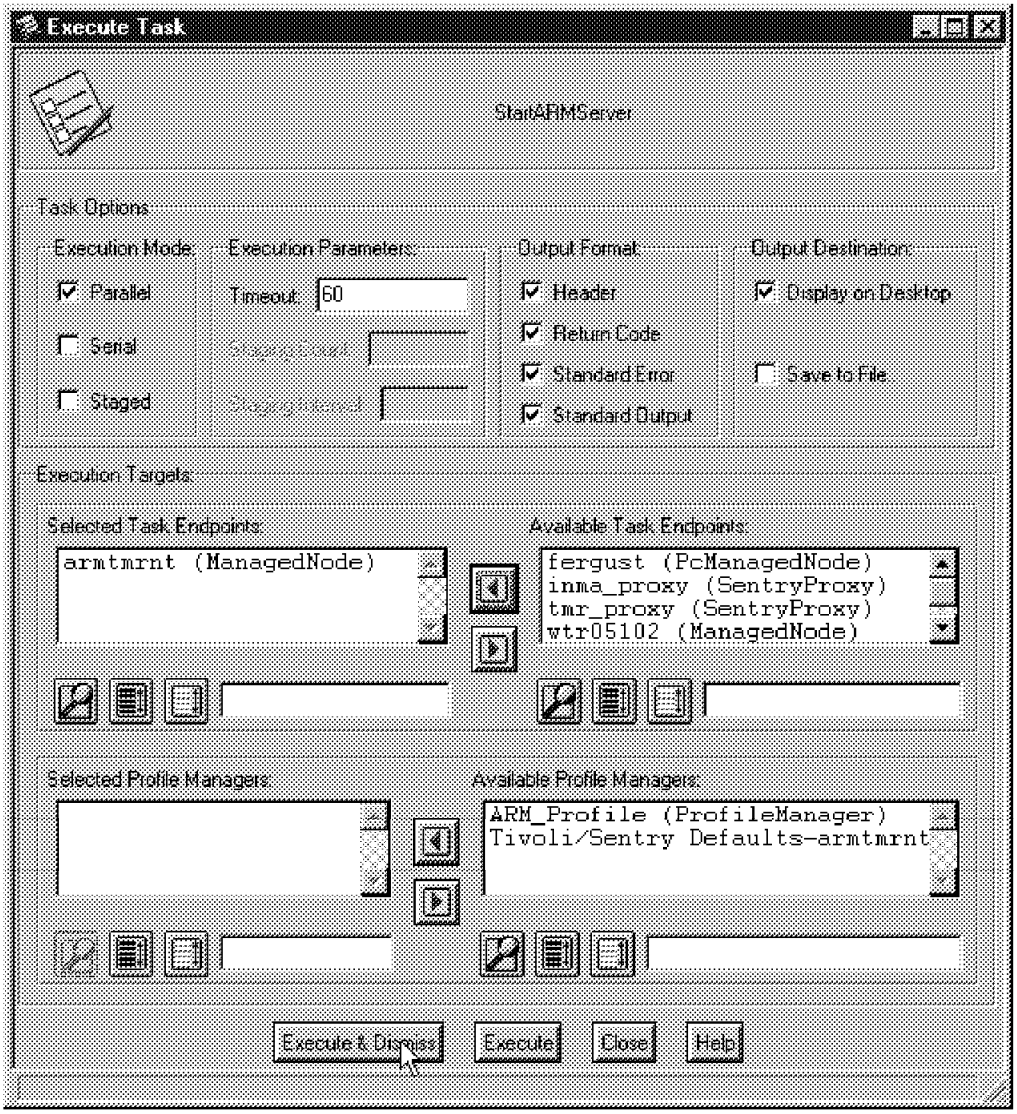


Figure 34. Selecting the Task Endpoint and the Profile Manager

A dialog box similar to Figure 35 on page 40 now appears. If you want to, you can specify the name of a log file that can later be sent to Tivoli Reporter, and whether you want to forward asynchronous events to Distributed Monitoring.

The log is optional, and at this point we did not use it. It is used to store data that can later be sent to Tivoli Reporter. See 5.1, “Setting Up the ARM Agents to Produce a Log File” on page 54 for details of how to do this.

We also chose not to send asynchronous events to Distributed Monitoring. So then we clicked on **Set & Execute**, as shown in Figure 35 on page 40.

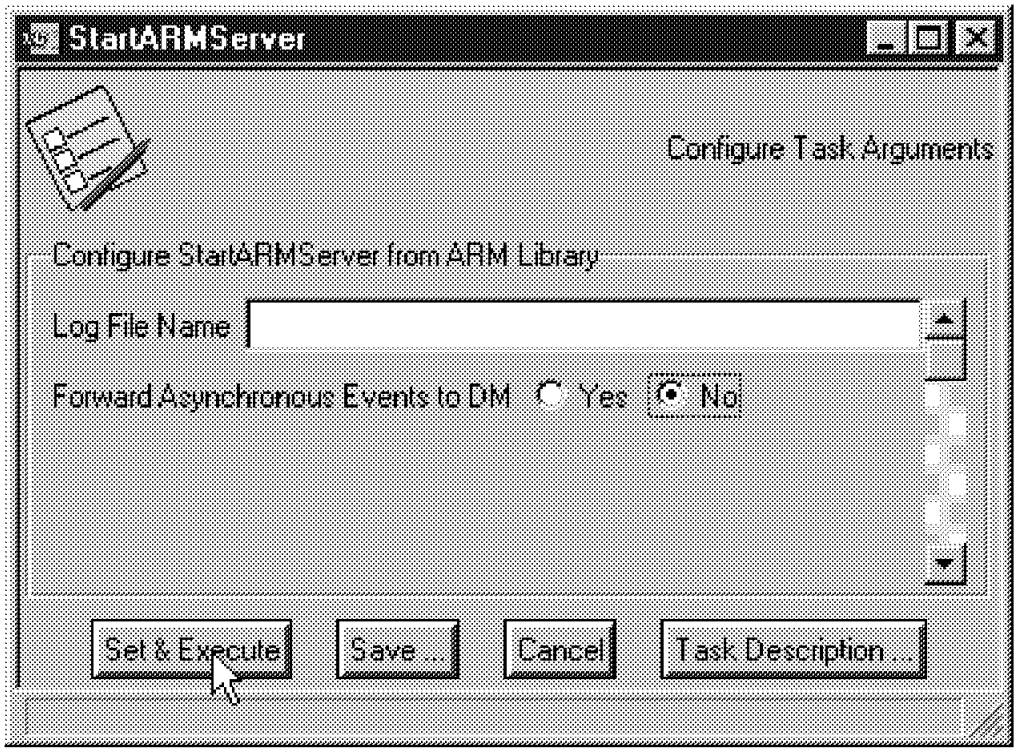


Figure 35. The Task Arguments Dialog Box

Figure 36 on page 41 displays a successful startup of the ARM Server Agent.

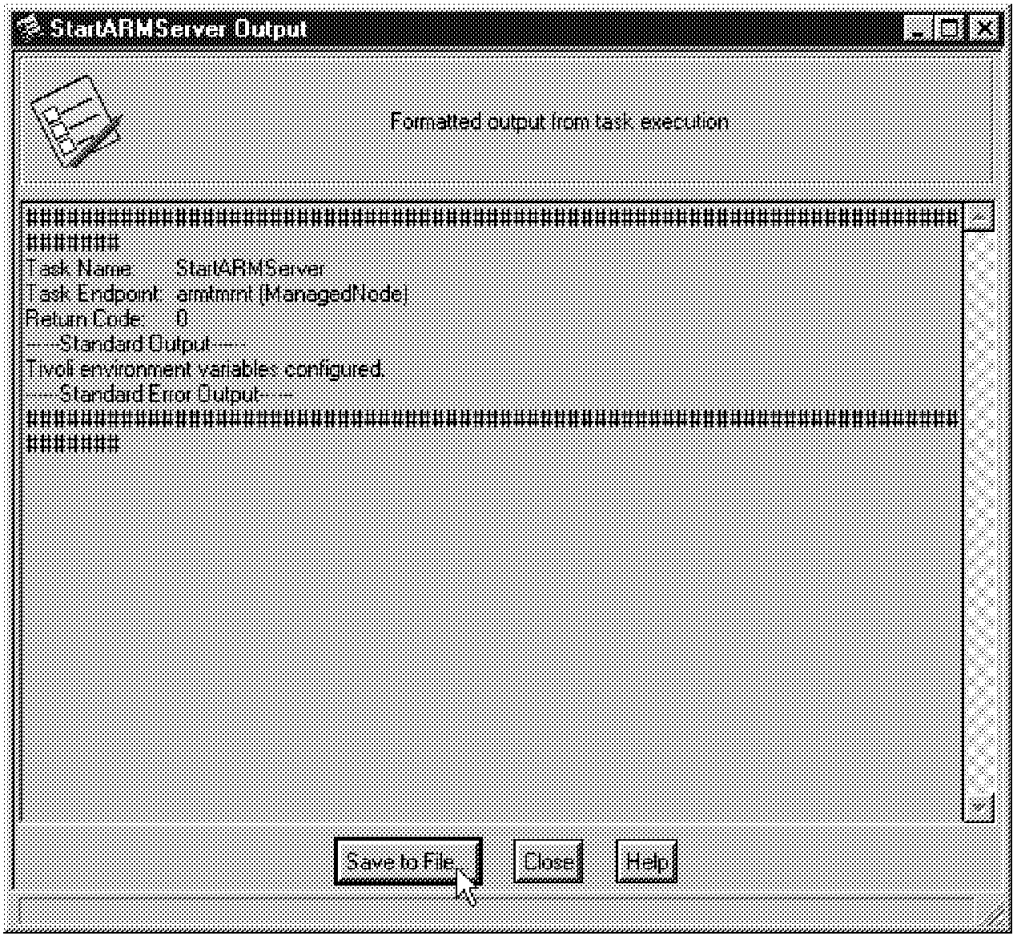


Figure 36. The ARM Server Agent Started Successfully

You may now want to check the status of the ARM Server Agent using the ServerStatus task from the ARM Library.

4.3 Starting the ARM Client Agent

Now that the ARM Server Agent is running, you can start an ARM Client Agent and connect it to the server.

4.3.1 Starting the ARM Client from the Command Line

To start the ARM Client Agent, type `warmcmd clntstart` at the command line on the ARM client machine.

The optional arguments are:

-h hostname Names the ARM server machine that you want this client to connect to. Before you start the client, make sure that the ARM Server Agent that you want to connect to is running.

We found that we could use either a host name or an IP address on this argument.

If you do not specify the -h argument, the client will attempt to connect to a ARM Server Agent on the same machine.

-f filename Allows you to specify a configuration file. See Chapter 7, “Production Use of the ARM Agents” on page 89 for details.

4.3.2 Checking the Client Status

We checked the status of the Client Agent and the contents of its log file.

If you issue the command `warmcmd clntstatus` from the command line on the machine where the ARM Client Agent is running, you should see responses similar to the following:

```
The ARM Client is active
Configuration file:
```

If you look in the message log for the ARM Client Agent, asxclnt, you should see messages similar to the following:

```
11/11/1997 17:17:16 - ASX0027I The ARM Client Subagent is running.
```

This is the log file for the ARM Client Agent, and the messages in the file can be very helpful in isolating any problems that you may encounter.

Note that the directory path in which this file resides will vary, depending on the platform and the Tivoli installation; rather than list them all here, we suggest that you use the Find command, to locate the file.

You can also use the netstat command to check that the client has connected to the ARM Server Agent on TCP port 2600. This is the default port; if you need to change it, follow the instructions in *TME 10 Distributed Monitoring ARM Agents User's Guide*.

```
C:\>netstat -a

Active Connections

  Proto  Local Address          Foreign Address        State

  TCP    wtr05165:1044          armtmrnt.itso.ral.ibm.com:2600  ESTABLISHED
```

4.3.3 Starting from the Tivoli Desktop

To start the ARM Client Agent from the TME Desktop, open the Policy Region for the ARM agents.

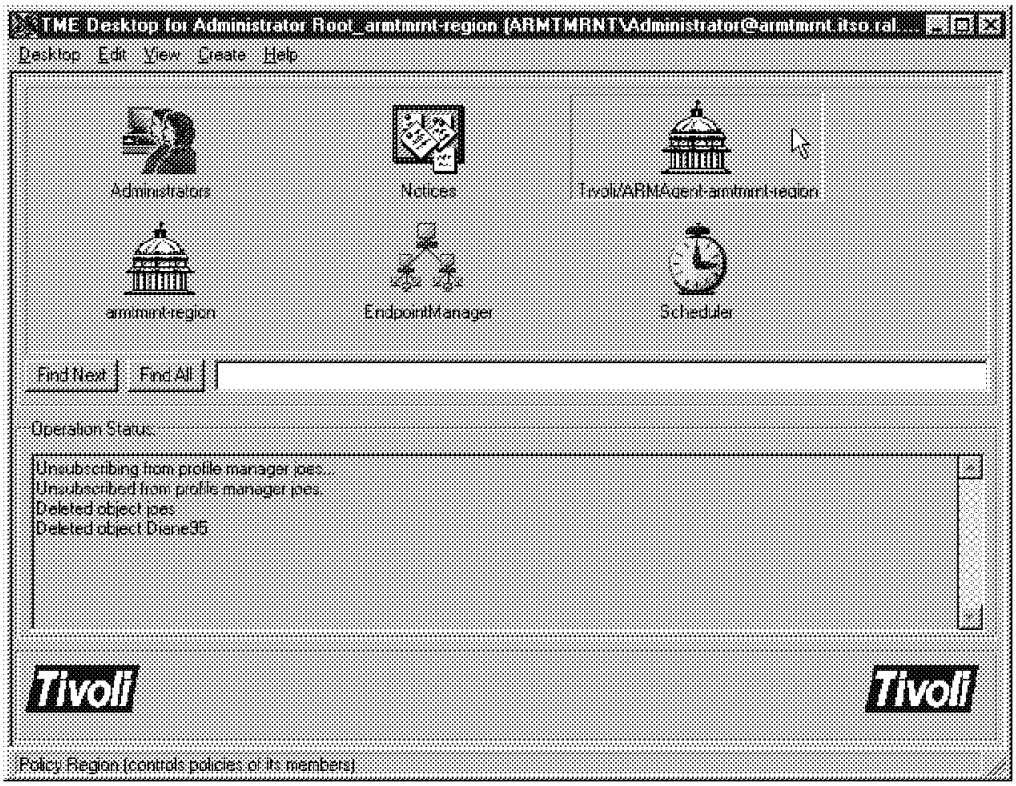


Figure 37. Opening the Policy Region for the ARM Agents

Now open the **ARM Library**.

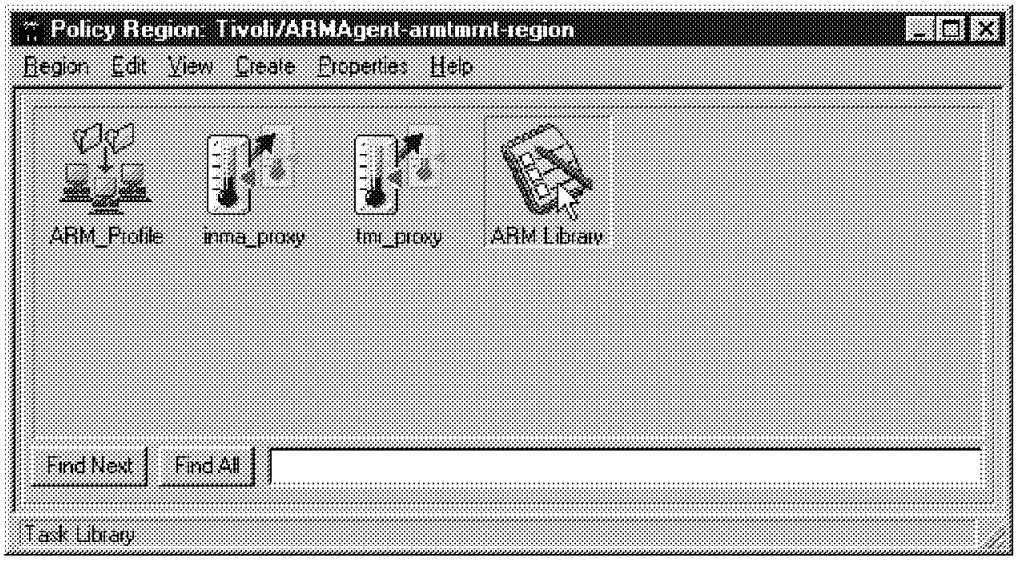


Figure 38. Opening the ARM Task Library

Note: If you want to execute the StartARMClient task on a Windows NT managed node, you must modify the execution privileges of the task. This can be done as follows:

1. In the ARM Library, click on the **StartARMClient** task with the right mouse button, and select **Edit Task**.

2. Enter * in the User Name field in the Edit Task dialog box that appears, as shown in Figure 39 on page 44.

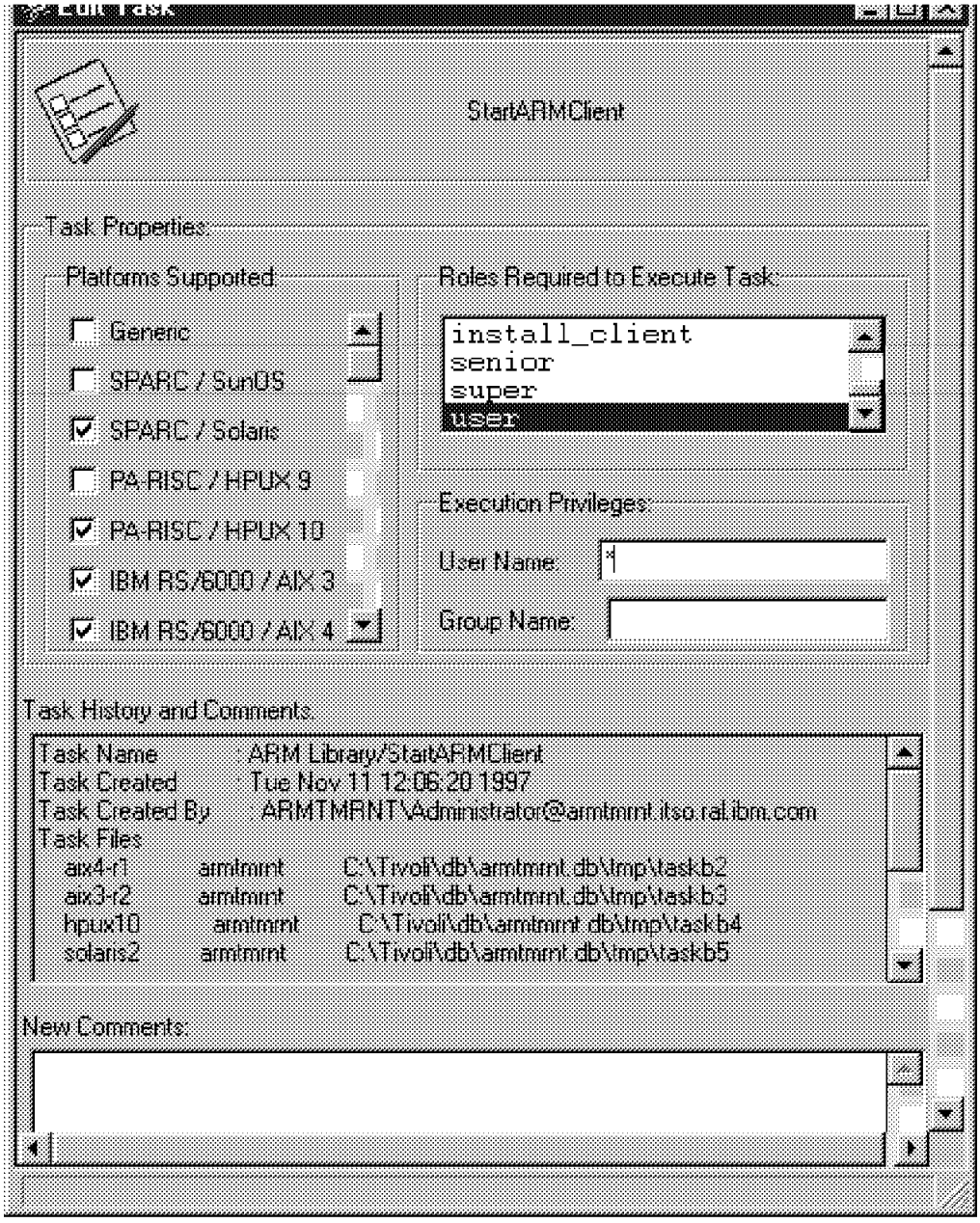


Figure 39. Setting the Execution Privileges for StartARMClient

Note that the user role is selected.

3. Click on **Change & Close** to close the dialog box.

Now open the **StartARMClient** task, and you should see a screen similar to Figure 40 on page 45.

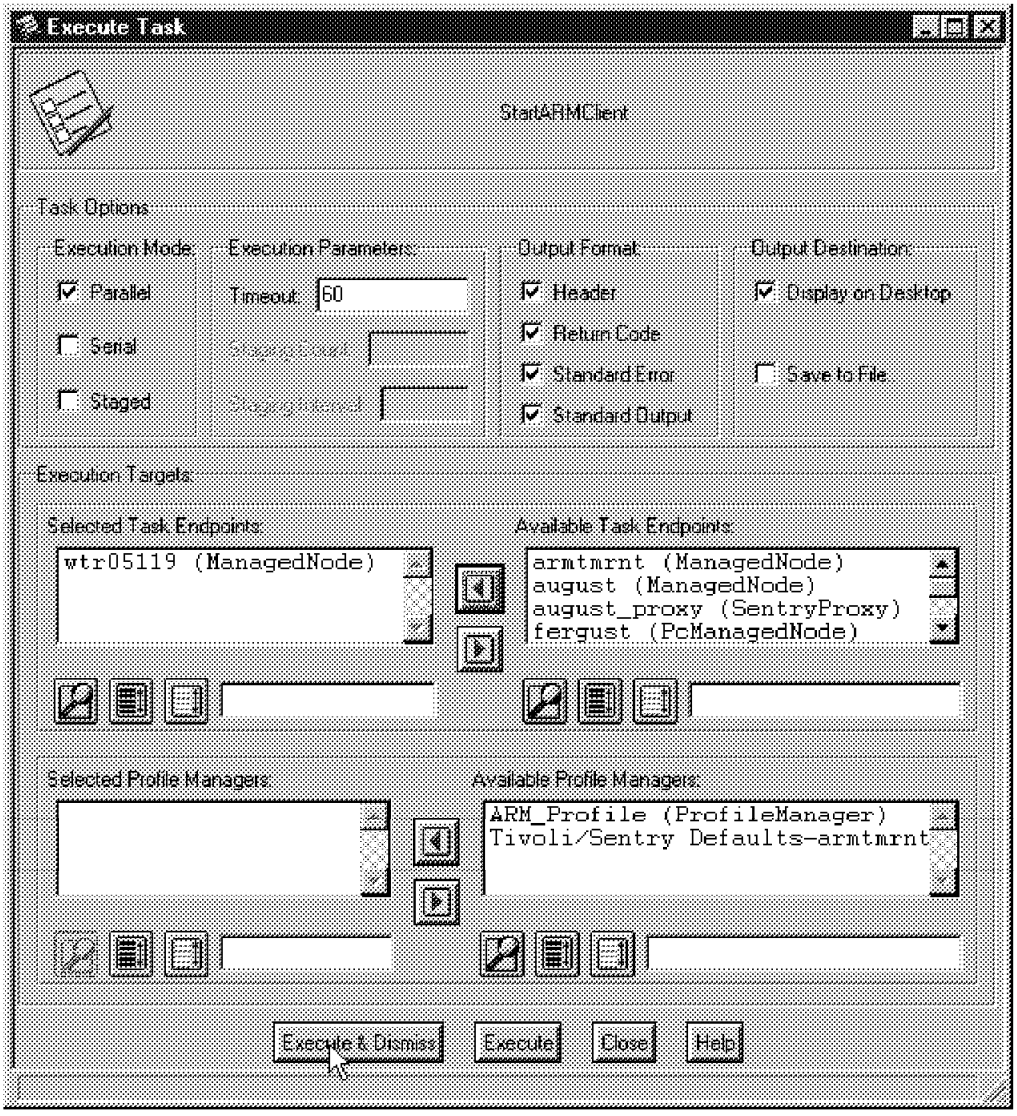


Figure 40. The StartARMClient Task

Select the **Task Endpoint** where you want to start the ARM Client Agent. It′s important here to select the resource, rather than its SentryProxy. Then check the **Display on Desktop** box, and click on **Execute & Dismiss**.

You will now be prompted to specify the name of the ARM Server Agent to which this ARM Client Agent should connect, and the name of a client configuration file.

Both of these fields are optional.

The Server Host Name field is used to specify the hostname or IP address of the ARM Server Agent that you would like this ARM Client Agent to connect to. If you leave it blank, the ARM Server Agent will be assumed to be on the same machine as the ARM Client Agent.

The Configuration File field can be used to specify the name of a file that contains configuration information for the ARM Client Agent. See Chapter 7, “Production Use of the ARM Agents” on page 89 for details.

Click on **Execute & Dismiss**, and you should receive messages similar to the following:

```
########################################
Task Name: StartARMClient
Task Endpoint: wtr05119 (ManagedNode)
Return Code: 0
------Standard Output------
------Standard Error Output------
########################################
```

You may now want to check the status of the ARM Client Agent using the ClientStatus task from the ARM Library.

4.4 Starting An ARM Collection

Now that the agents are running, you can decide which ARM-instrumented applications you want to collect information on. This is done by issuing a command, or executing a task, on the ARM Client Agent.

4.4.1 Starting Collection from the Command Line

To start the collection, we issued the following command on the ARM client machine:

```
warmcmd startcoll -a abcd -t efgh -i 60
```

We used an application name of *abcd*, and a transaction name of *efgh*. The *-i* flag is used to specify the collection interval.

Case sensitive

The application and transaction names are unfortunately case sensitive, so make sure that you get the case correct when starting collection on an application/transaction pair, and when requesting collected data.

The following message appears on the client machine when the collection is started:

```
The collection was started.
```

4.4.2 Starting Collection from the Tivoli Desktop

Open the **StartCollection** task from ARM policy region's task library.

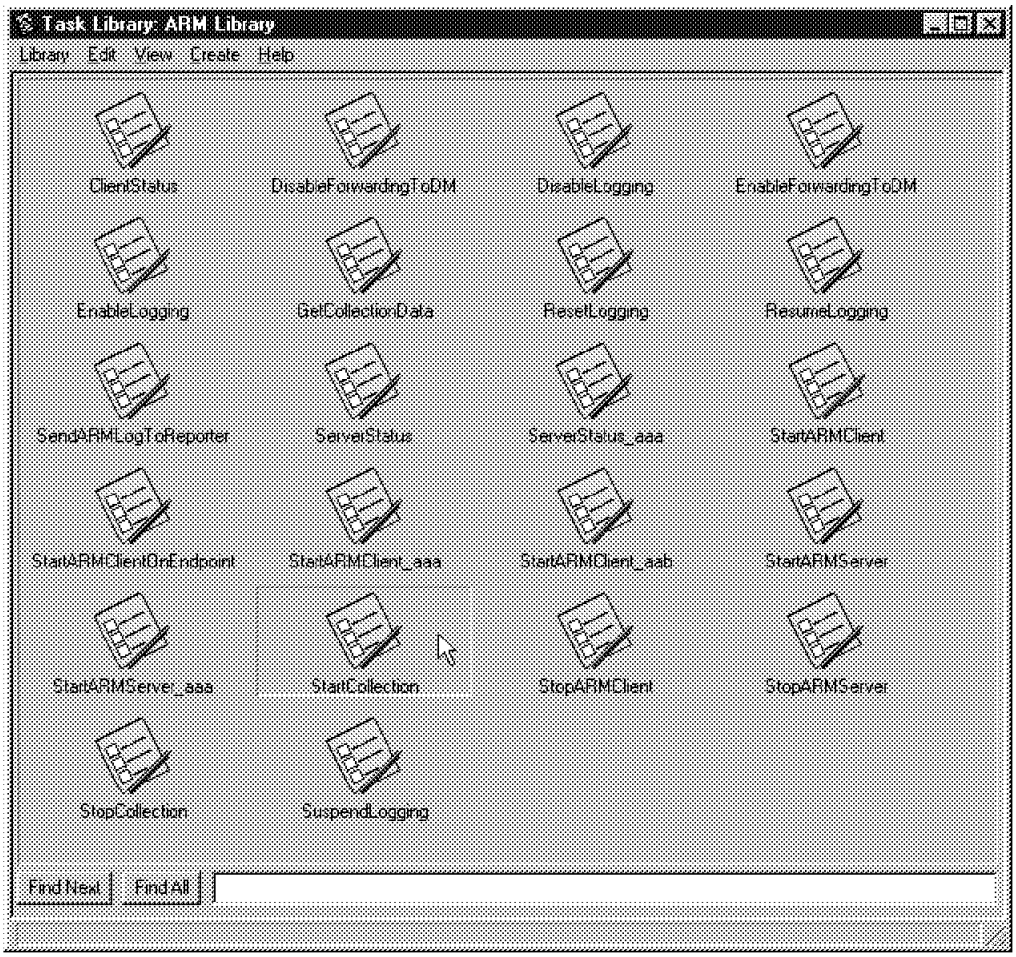


Figure 41. Opening the StartCollection Task

Now select the Task Endpoints where you want collection to start, from the Available Task Endpoints, and the profile managers from the Available Profile Managers.

You should check **Display on Desktop**, to see that the collections start properly.

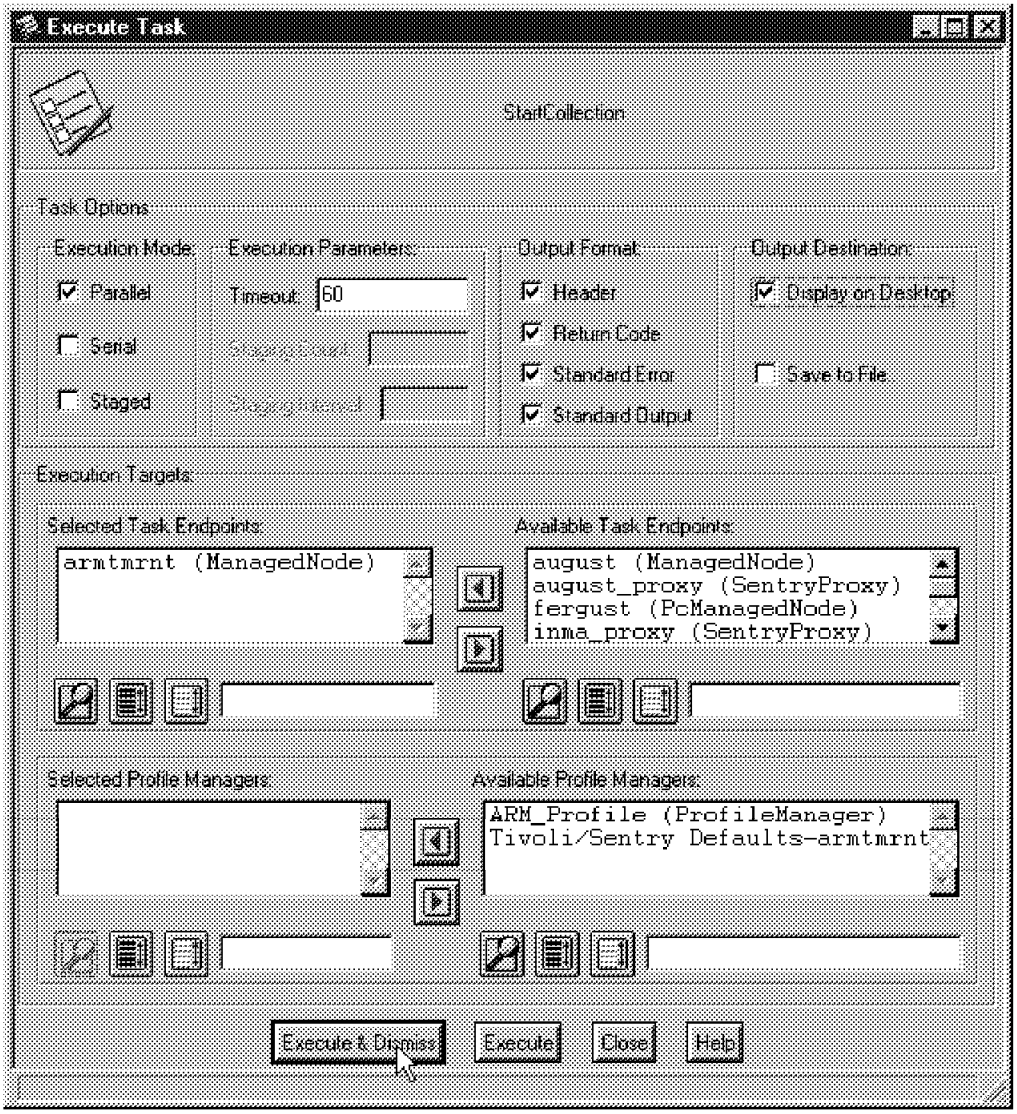


Figure 42. Selecting the Task Endpoint

The Application Name, and Transaction Name fields are mandatory.

The Application Userid field is supposed to be optional, but in fact you must specify *""* in that field, which will give you all the user IDs for any matches to the Application Name and Transaction Name you specified. Unfortunately, you cannot enter a wildcard for the Application Name, or Transaction Name.

The Interval (seconds) field has a default of 360. This can be changed to meet your individual needs, but although it will accept any number, it will not start a collection with less than a 60-second interval.

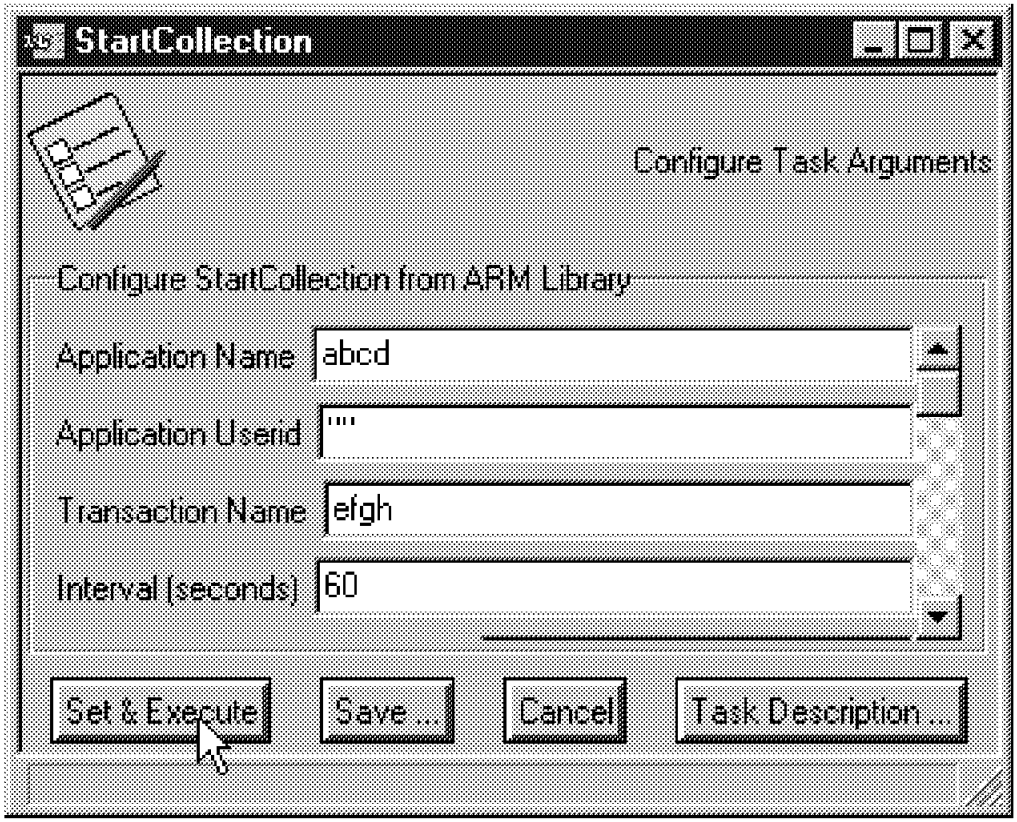


Figure 43. Setting the Task Arguments

You should now see messages similar to the following:

```
##########################################
Task Name: StartCollection
Task Endpoint: armtmrnt (ManagedNode)
Return Code: 0
------Standard Output------
------Standard Error Output------
##########################################
```

4.5 Testing the Agents With the ARMTEST Program

Now that you have the agents installed and running, and collection is running, we recommend that you use our instrumented application armtest to check that everything is working properly.

See Appendix B, “How to Get the Samples Used in this Book” on page 121 for instructions on how to get the armtest program from the Internet. Make sure that you use the correct executable for the platform where your ARM Client Agent is running.

You can see the source of the armtest program in Appendix A, “Source File Listings” on page 115.

When you run the armtest program, you will be prompted to enter names for the application and the transaction that it will create. these names must match the names that you used when you started collection in 4.4, “Starting An ARM Collection” on page 46.

You should see a dialog box similar to the one shown here.

```
C:\>armtest

This program generates transactions for the ARM Agents of
Distributed Monitoring, on Win 95 and NT workstation.

Please enter the name of your application:
abcd
Your application handle is: 1
Your application name is: abcd

Please enter the name of your transaction:
efgh
Your transaction handle is: 1
Your transaction name is: efgh

Type "start" to start the transaction
start

Transaction started...
Your start handle is: 1
Type a few characters and press ENTER.

Or type "end" and press ENTER, to end the loop.
end


Transaction stopped.
Your stop return code is: 0


Application ended.
Your end return code is: 0

C:\>
```

4.6 Displaying Data in Text Form

At the end of a collection interval, the ARM Server Agent receives data from the ARM Client Agent, and stores it in a buffer. When the next interval expires, the buffer is cleared and new data from the ARM Client Agent is written into it.

You can issue the warmcmd getdata command on the ARM Server Agent machine to display the contents of this buffer, as you can see below.

```
C:\>warmcmd getdata -a abcd -t efgh -c wtr05119 -v
Transaction: abcd..efgh
Client:      wtr05119
Maximum Response Time:       13.409
Minimum Response Time:        0.231
Average Response Time:        0.902
RT Standard Deviation:        2.208
Transaction Count    :           62
Failed  Transactions :            0
Aborted Transactions :            0
Bucket #1 Count      :           62
Bucket #2 Count      :            0
Bucket #3 Count      :            0
Bucket #4 Count      :            0
Bucket #5 Count      :            0
Bucket #6 Count      :            0
Bucket #7 Count      :            0
```

Case sensitive

The application and transaction names are unfortunately case sensitive, so make sure that you get the case correct when starting collection on an application/transaction pair, and when requesting collected data.

Note: If you issue the warmcmd getdata command on a ARM Client Agent, you will get the message `ARM Agent is not active`, which is rather misleading.

The GetCollectionData task can also be used, instead of the warmcmd getdata command.

Unfortunately, there is no way to get data for all users by issuing just one warmcmd getdata command.

If you specify * as the user ID, you will receive data about the first collection whose application and transaction names match the arguments specified, regardless of the user ID.

If you are interested in all users you must issue a warmcmd getdata command for each of them.

Chapter 5. Reporting ARM Data with Reporter

Reporter provides very powerful functions to summarize large volumes of data and present them in a variety of easy-to-read graphical formats.

When the ARM data arrives at the ARM Server Agent it can be recorded in a log file that can later be sent to Reporter.

This process is shown in Figure 44.

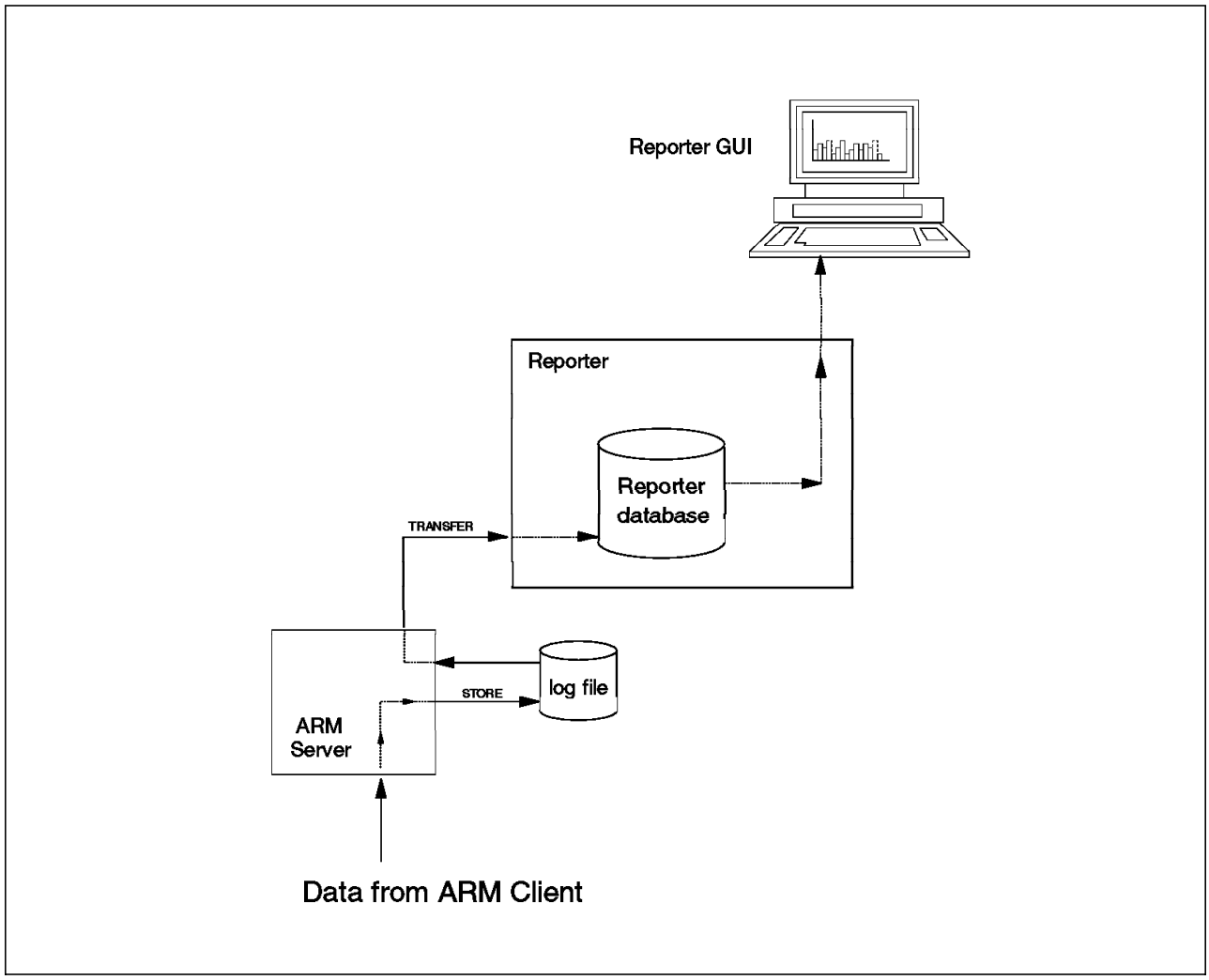


Figure 44. Processing ARM Data with Reporter

We had installed Reporter, as in Appendix C, “Installing Reporter” on page 123. In order to report ARM data, we completed the following steps:

1. Set up the ARM agents to write into log files
2. Send the log files from the ARM Server Agents to Reporter
3. Set up Reporter to import the ARM log files
4. Run reports of the ARM data

5.1 Setting Up the ARM Agents to Produce a Log File

In order to create a log file, the ARM Server Agent must be started with the *-f filename* flag, to create the file, and at least one collection must be started on the ARM Client Agent.

5.1.1 Starting the ARM Server Agent

On the machine where we wanted to start the ARM Server Agent we typed the command:

```
warmcmd srvstart -f c:\log\fred
```

This starts the server and creates a file called *fred* in the *log* directory.

If you don't specify a complete path, this file is created in the `Tivolidbarmtmrnt.dbtmp` directory.

One record is appended to the file for each ARM Client Agent, after the expiration of each collection interval. We found that the size of a record in this file was 247 bytes.

We issued the command:

```
warmcmd srvstatus -v3
```

to check the status of the ARM Server Agent and got the following messages:

```
The ARM server subagent is active
Data logging enabled on file: c:\log\fred
Forward asynchronous events to DM: disabled
```

This specifies the file in which our data is collected.

5.1.2 Starting the ARM Client Agent

On the ARM Client Agent machine, we issued the command:

```
warmcmd clntstart -h armtmrnt
```

to start the ARM Client Agent, connecting it to the ARM Server Agent on the machine *armtmrnt*.

5.1.3 Starting Collection on the ARM Client Agent

On the ARM Client Agent machine, we issued the command:

```
warmcmd startcoll -a abcd -t efgh -i 60 -f
```

to start collection on the ARM Client Agent.

The flags are as follows:

-a abcd	Starts collection on application abcd.
-t efgh	Starts collection on transaction efgh.
-i 60	Data is collected every 60 seconds.
-f	This initiates the data collection in the file that was specified when the server was started. Note that the log file name does NOT need to be specified here.

5.2 Sending the Log File to Reporter

Now that we had some records in the log file, we wanted to send it over to Reporter. This is done by invoking an FTP service, so it is *not* necessary for the Reporter machine to be in the same TMR as the ARM Server Agent.

The FTP service that transfers ARM log files to Reporter is invoked from a task in the ARM library. By contrast, almost all the other performance agents that can send data to Reporter do so under Reporter control.

Note that the SendARMLogToReporter task deletes the log file from the ARM Server Agent, unless it does not have sufficient privileges to do so.

5.2.1 Executing the Task

This function cannot easily be performed from the command line, so start by executing the `SendARMLogToReporter` task from the ARM task library, as shown in Figure 45.

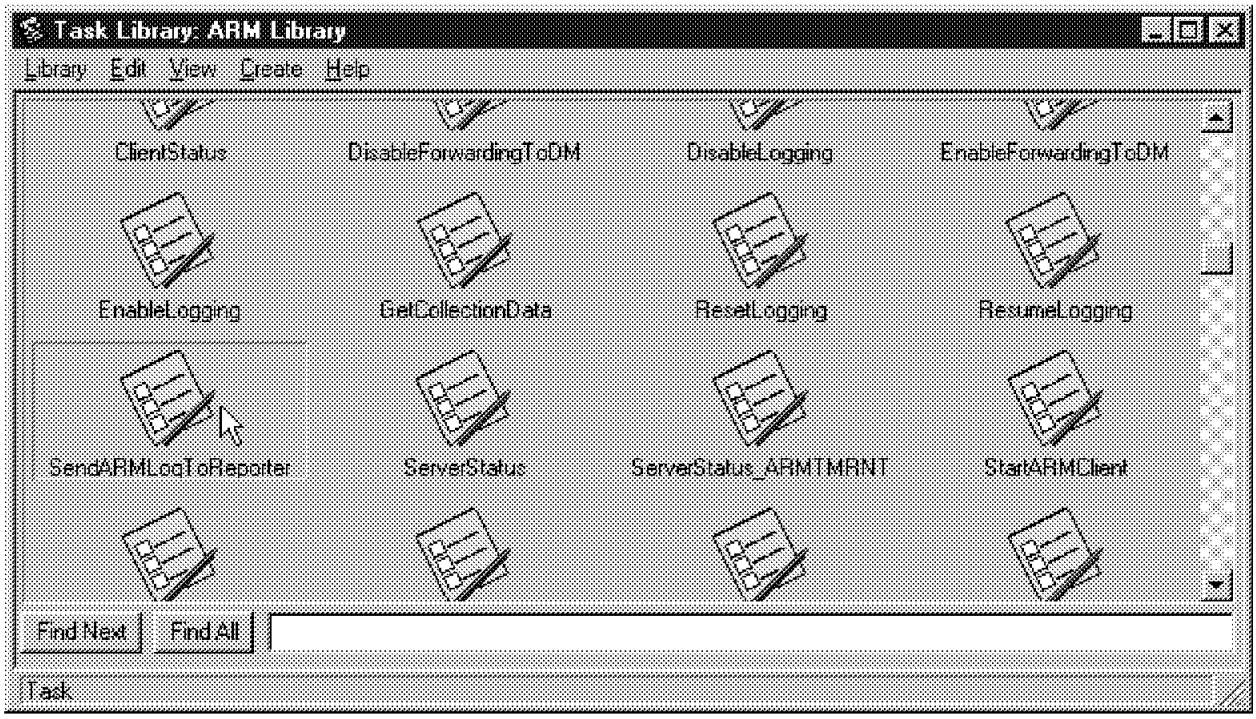


Figure 45. Executing the SendARMLogToReporter Task

You will now see a dialog box similar to Figure 46 on page 56.

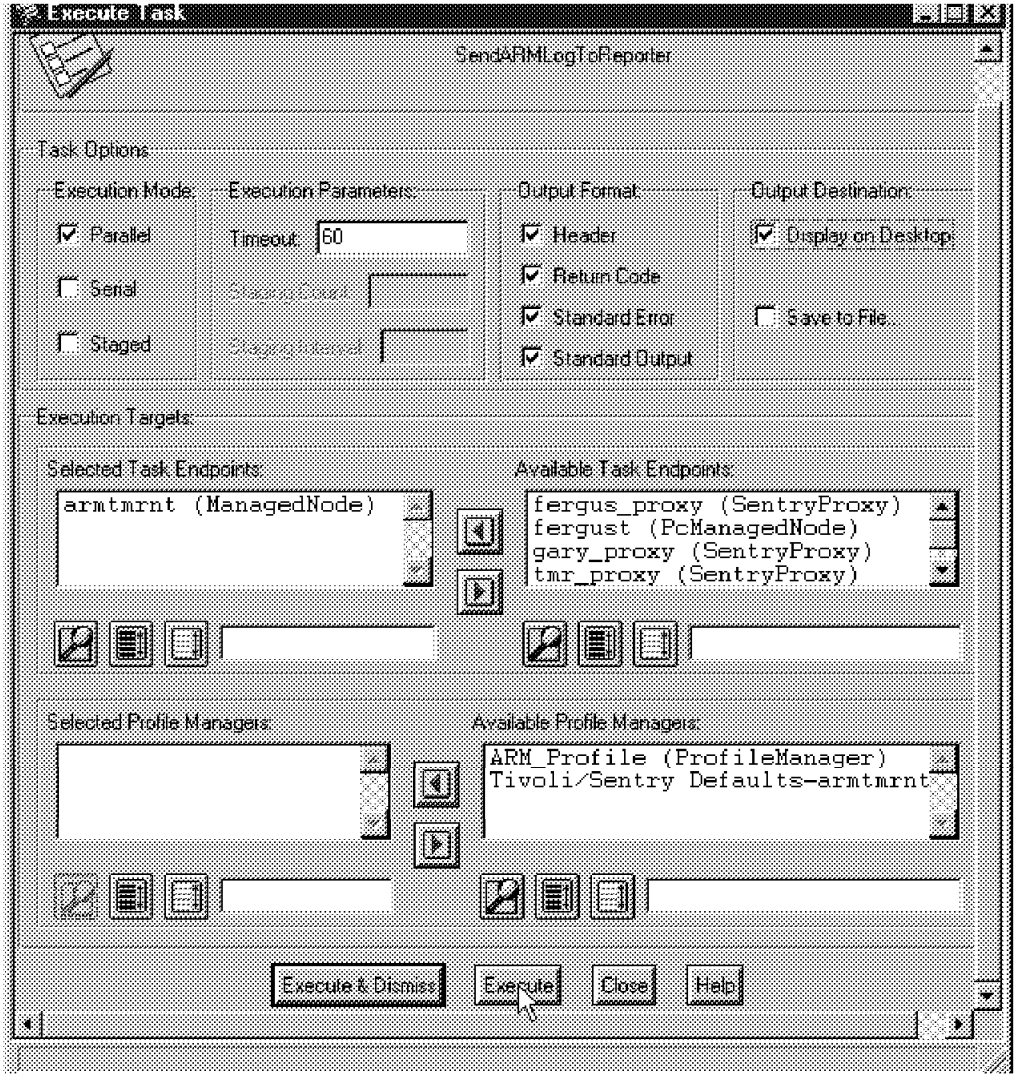


Figure 46. Execution Parameters for the SendARMLogToReporter Task

You must set the Task EndPoint to the Managed Node where the ARM Server Agent that holds the log file resides. We suggest that you check the **Display on Desktop** box, so that you see the messages indicating the success or failure of the task when it runs.

Now click on **Execute** and you will be presented with a dialog box similar to Figure 47 on page 57.

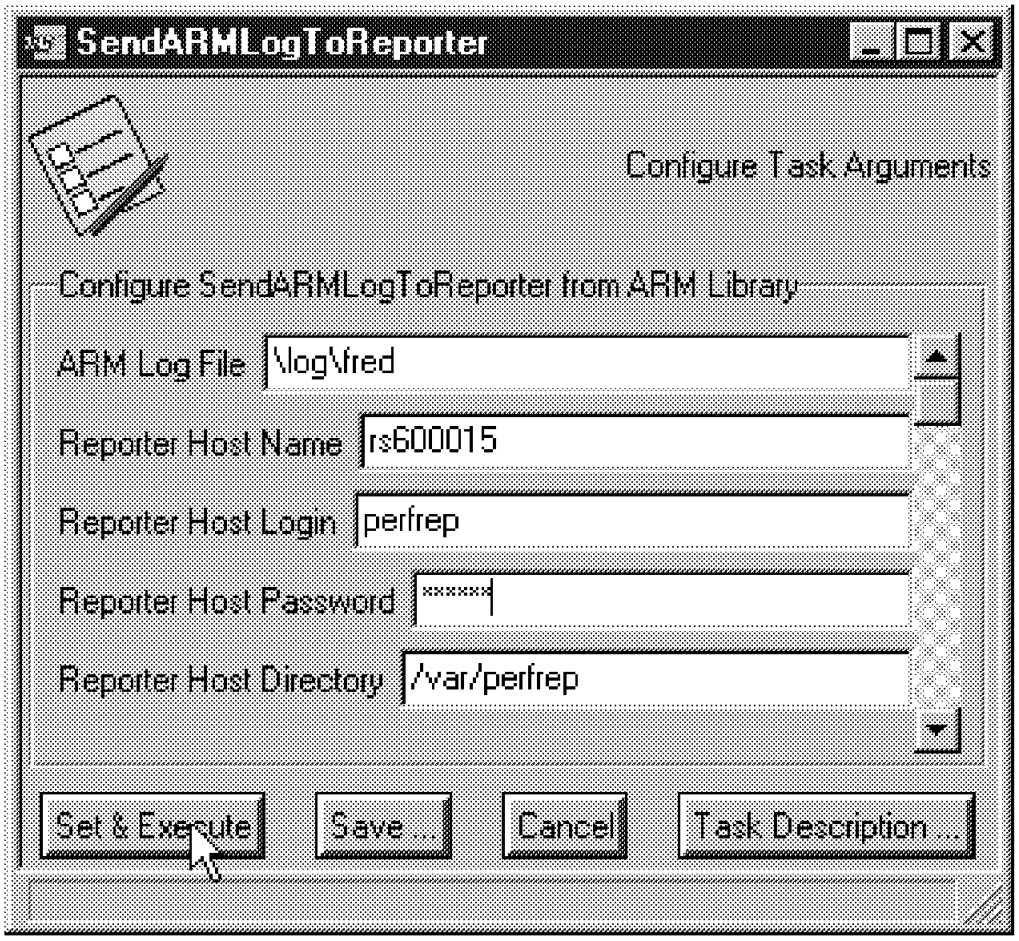


Figure 47. Supplying Details of the Reporter Target

Fill in the following details:

- The full path and name of the log file on the ARM Server Agent machine
- The hostname or IP address of the machine where Reporter resides
- The user ID of the Reporter Administrator.

  We had created an ID called *perfrep* in Appendix C, “Installing Reporter” on page 123. Since Reporter was running on AIX, this name is case sensitive.
- The password for the user ID above. Since Reporter was running on AIX, this name is also case sensitive.
- The target directory on the machine where Reporter is installed. The default directory is `/var/perfrep`. This name is case sensitive. The log file will be FTPed into this directory.

These parameters reflect our environment shown in Figure 48 on page 58.

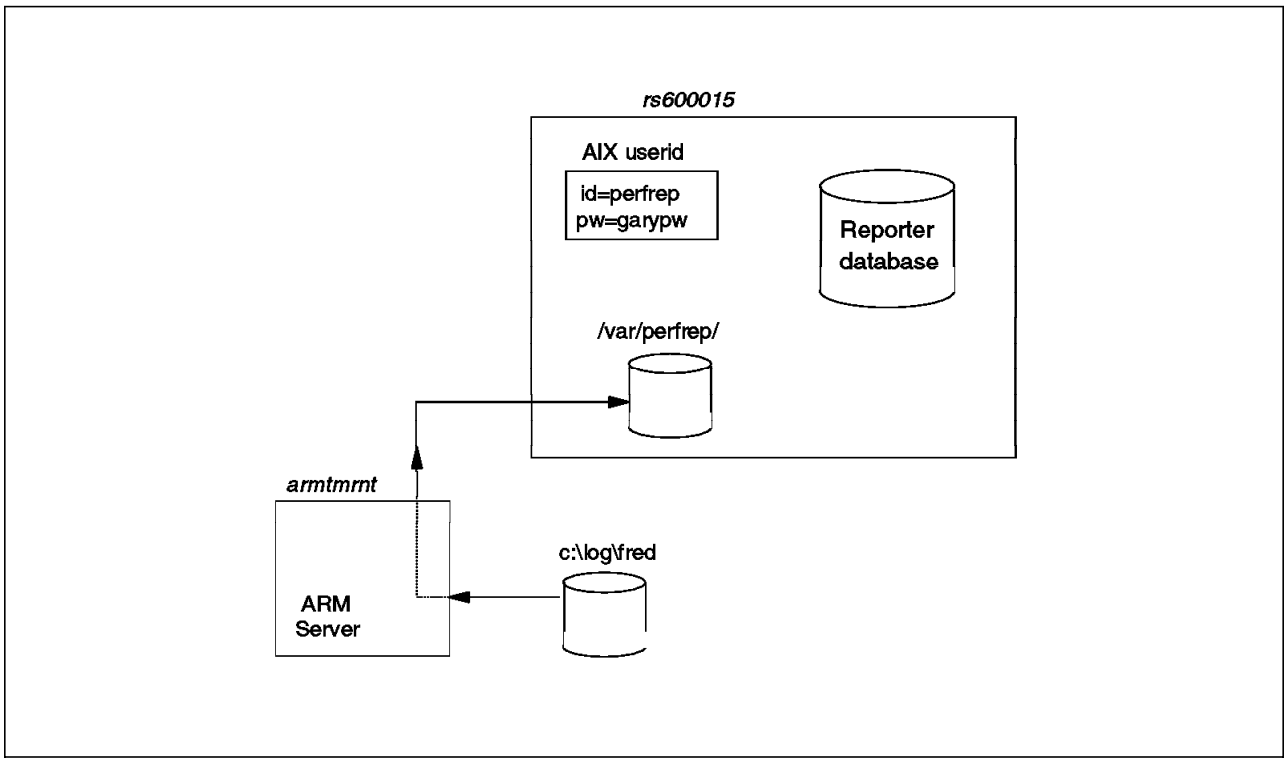


Figure 48. The Environment for the SendARMLogToReporter Task

When we clicked on **Set & Execute**, a parameters file *ftp_ini* was created in the Tivolidbarmtmrnt.dbtmp directory with the following contents:

```
open rs600015
perfrep
garypw
cd /var/perfrep
lcd C:\Tivoli\db\armtmrnt.db\tmp
put armtmrnt.applmon.d980312.1
bye
```

This is then used by the task to perform the file transfer.

Upon completion of the task, we received the following messages indicating that the file transfer completed successfully.

```
###########################################
Task Name: SendARMLogToReporter
Task Endpoint: armtmrnt (ManagedNode)
Return Code: 0
------Standard Output------
Output filename: armtmrnt.applmon.d980312.1
------Standard Error Output------
###########################################
```

When we looked in the /var/perfrep directory on our rs6000, we could see a new file called armtmrnt.applmon.d980312.1.

This file fits the naming convention for all files sent to Reporter, which is:

hostname.logname.d**date.**1

where:

- *hostname* is the hostname of the ARM Server Agent that sent the file.
- *logname* is the name of the log. ARM logs are always called *applmon*.
- *date* is the day before the creation of the log file.

Reporter expects you to create the log file early in the morning, storing the data for the previous day, so when the SendARMLogToReporter task runs, it calculates yesterday's date and makes this the date part of the filename.

We ran the SendARMLogToReporter task on March 13th, so the file has 980312 in the date part of its name.

By default, Reporter starts importing files into its database at 3 a.m., so your files will need to be on the Reporter machine before that time. Therefore, we recommend using the Scheduler service on the Tivoli Desktop to schedule the SendARMLogToReporter task, as shown in 7.5, "Automating the Transfer of Log Data to Reporter" on page 92.

5.3 Setting up Reporter to Import ARM Log Files

Now that you have the log file on the Reporter machine, its data must be imported into ARM-specific tables in the Reporter database.

5.3.1 Before You Start

There is a bug in the Reporter code for ARM.

We received an early copy of a fix, which required that we replace the files *pramrt.rec* and *pramrt.tab* which were in the /usr/local/Tivoli/bin/aix4-r1/perfrep/def directory.

You should call service and ask for this fix. It must be installed before you install the Reporter component for ARM.

We also found that the temporary database used by Sybase, *tempdb*, was filled when we tried to install the ApplicationMonitor component for ARM. We issued the following commands to prevent this:

```
$ isql -Usa -P

1> dump transaction tempdb with no_log
2> go
1> sp_dboption tempdb, "trunc log on chkpt", true
2> go
Database option 'trunc log on chkpt' turned ON for database 'tempdb'.
Run the CHECKPOINT command in the database that was changed.
(return status = 0)
1> use tempdb
2> go
1> CHECKPOINT
2> go
1> quit
```

5.3.2 Installing the Reporter Component for ARM

In order to work with ARM data, you will need to install the Reporter component called ApplicationMonitor. This consists of ″log collector″ definitions, tables and reports specific to ARM data.

To do this, complete the following steps:

1. Log in as *perfrep*.

   Issue the command `echo $SYBASE`, to ensure that you have the environment variables set; they may not be set if you used su to switch user to *perfrep*.

2. Start reporter SMIT by typing `smitty perfrep`.
3. Select **Install and configure**.
4. Select **Install components**.
5. Then select **Install component**.
6. Press PF4 to list the components, then select **ApplicationMonitor**.

   Now your screen should look similar to this:

```
                              Install component

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Component name                                     [ApplicationMonitor]
* Table space name                                   [USERS]
  Table prefix                                        PERFREP
* Administrator                                       pradm
* List of users                                      [pruser]
```

7. Press Enter and the installation should proceed with messages similar to the following:

```
Checking the component ApplicationMonitor, please wait.

chmod: /var/perfrep/prinstarm.log: Operation not permitted.

Installing the component ApplicationMonitor, please wait.

prinstcomp: The component ApplicationMonitor is successfully installed.

Press Enter to exit, or type any nonblank character to display the log
file /var/perfrep/prinstarm.log
```

   You can review the installation messages in the file /var/perfrep/prinstarm.log.

5.3.3 Activating the Reporter Component for ARM on the ARM Server Agents

Now that the component for ARM is installed on the Reporter Manager machine, you will need to activate it on all the ARM Server Agent machines that will be sending log files to Reporter.

This can be performed before or after the log files have been sent to Reporter, but it *must* be completed before 5.3.4, “Setting Up Reporter Data Collection for ARM” on page 62.

Complete the following steps to do this:

1. Define a node group

   Log on as the user ID *perfrep* and run `smitty perfrep`.

   Select **Install and configure** and then **Administer nodes** followed by **Create a node group**.

   Choose a meaningful name for the new node group - we called ours *armgroup* - and then press Enter to create it.

2. Define every ARM Server Agent machine as a node in the node group.

   Now return to the Administer Nodes panel and select **Create a node**.

   Fill in the fields, so that your screen looks something similar to this:

```
                                Create node

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Node name                                          [armtmrnt]
* Accounting group                                   [armgroup]
* Domain name                                        [itso.ral.ibm.com]
* IP address                                         [9.24.104.215]
  Description                                        []
```

   This needs to be repeated for every node that might send data to Reporter.

3. Activate the Reporter component for ARM on all of these nodes.

   From the Install and configure menu, select **Activate components**. This will take you to the Activate components menu.

   Here you can activate the ApplicationMonitor component on individual nodes, one at a time, or on a node group that contains the nodes.

   We acivated our first node individually, by selecting the **Activate component on node** option, and then filling in the fields as you can see below.

```
                          Activate component on node

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Node name                                          [armtmrnt      ]
* Component                                          [ApplicationMonitor]
```

   Now you should see messages similar to the following:

```
                                COMMAND STATUS

Command: OK            stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

rs600015 1998/04/17 10.02.17 PRADM021I  pradmcoll: The log APPLMON is
activated on the node armtmrnt.
```

This needs to be repeated for every node that might send data to Reporter.

5.3.4 Setting Up Reporter Data Collection for ARM

Reporter uses the term *Data Collection* in a rather confusing way.

Reporter Data Collection involves two processes:

1. Log files are transferred from Reporter agents to the Reporter Manager.

   In the case of the ARM agents, the file transfer is performed by the SendARMLogToReporter task on each ARM Server Agent, as we saw in 5.2, “Sending the Log File to Reporter” on page 55, so this part of the Data Collection process does not need to be performed from Reporter.

2. The log files are summarized and stored in the Reporter database.

   The process of summarizing the log data and storing it in the Reporter database is unfortunately called *Collection*, and is performed by the Reporter *Log Collector* function.

All of this is shown in Figure 49.

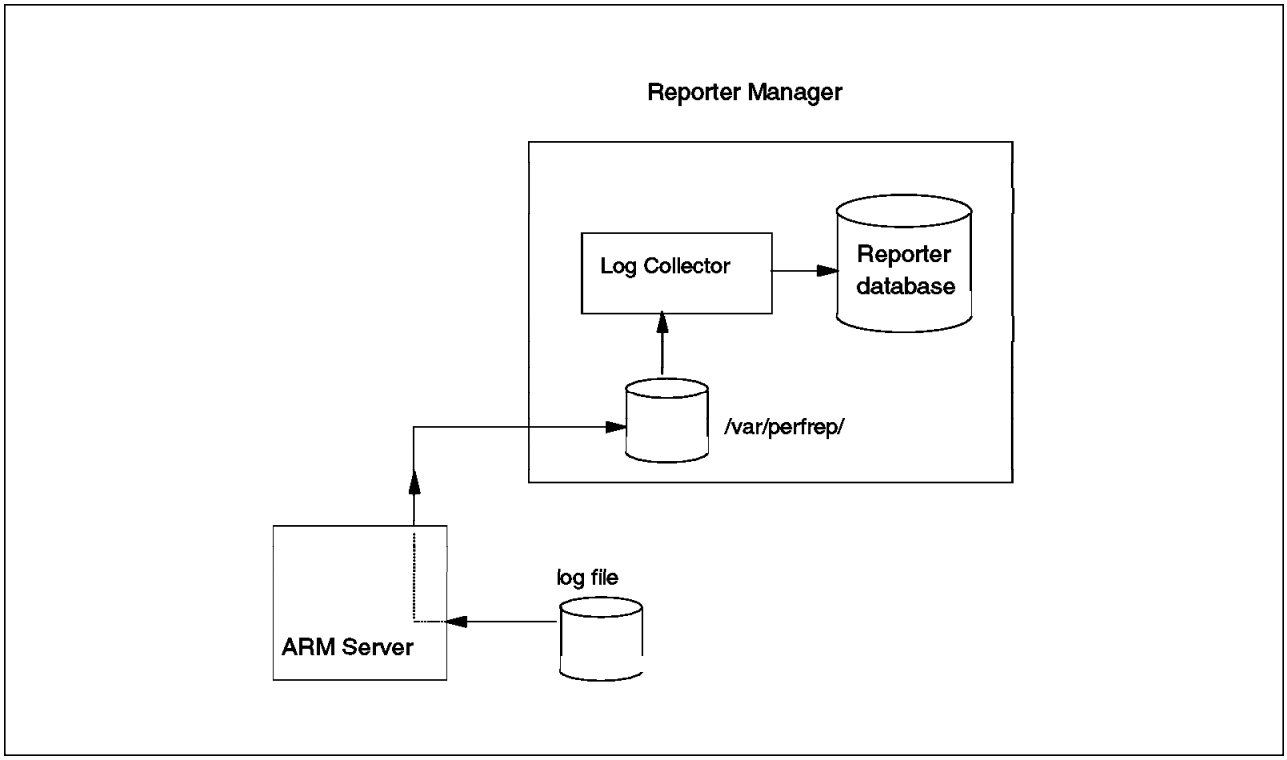


Figure 49. Reporter Data Collection for ARM

5.3.4.1 Starting Collection

There are two ways that you can use Reporter data collection:

- Run a single data collection
- Start Data Collection

We recommend that you first run a single data collection, to see that everything is working OK, and then start (regular) data collection.

To start a single data collection, run smitty perfrep, select **Operate** and then **Collect** then **Run single data collection**.

Now your screen should look similar to this:

```
                          Run single data collection

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [ntry Fields]
  Action                                             [Transfer & Collect]
  Date of log data                                   [980416]

  Node group                                         [All]
  or
  Node name                                          []

  Component                                          [All]
  or
  Log name                                           []
  Purge                                              [Yes]
  Buffer size                                        []
```

Since the ARM log files are transferred to Reporter by the SendARMLogToReporter job on the ARM Server Agent, you must change the Action to Collect only.

We set the Node to armtmrnt and the Component to ApplicationMonitor.

When we pressed Enter, we saw messages confirming that the collection process had been completed.

Now you know that the collection process for ARM works correctly, you should set it up so that it runs automatically every day at 3 a.m., like every other Reporter collection.

Make sure that you are logged in as the *perfrep* user, and then run smitty perfrep, select **Operate** and then **Collect** then **Start data collection**.

You will be prompted for confirmation, and then the collection will be scheduled in cron, so if you look in the crontab for the *perfrep* user, you should see an entry similar to the following:

```
0 3 * * * /usr/lpp/perfrep/bin/prcoll -p/home/perfrep/etc/perfrep.cfg
```

This command will run at 3 a.m. every morning, and run the *prcoll* script. This script will import any files in the /var/perfrep directory into the Reporter

database, provided that they have yesterday's date as a part of the filename, as shown in 5.2.1, "Executing the Task" on page 55.

You should see output in the file /var/perfrep/prcoll.log similar to the following:

```
rs600015 1998/04/17 11.10.52 PRADM009I  prcoll: The log APPLMON for
the date 980416 was transferred and collected on the nodes: armtmrnt.
```

And you should see output in the file /var/perfrep/prcoll.prlogcol similar to the following:

```
collect APPLMON from '/var/perfrep/armtmrnt.applmon.d980416.1'
PRCOL300I Collect started at 1998-04-17-03.00.00.
PRCOL302I Processing /var/perfrep/armtmrnt.applmon.d980416.1.
PRCOL341I The first-record timestamp is 1998-04-16-10.49.00.000000.
PRCOL328I Reading lookup table PERFREP.NODE: 2 rows, 48 bytes.
PRCOL328I Reading lookup table PERFREP.CAL_SPECIAL_DAY: 2 rows, 76 bytes.
PRCOL328I Reading lookup table PERFREP.CAL_DAY_OF_WEEK: 7 rows, 210 bytes.
PRCOL328I Reading lookup table PERFREP.CAL_DAY_PERIOD: 18 rows, 972 bytes.
PRCOL342I The last-record timestamp is 1998-04-16-10.49.00.000000.
PRCOL310I A database update started after 1 records due to end of log.
PRCOL003I
PRCOL315I Records read from the log or built by log procedure:
PRCOL317I Record name        |    Number
PRCOL319I AM_RT              |         1
PRCOL321I Total              |         1
PRCOL003I
PRCOL323I                              -------Buffer------ ------Database-----
PRCOL324I Table name                  |   Inserts   Updates   Inserts   Updates
PRCOL325I ----------------------------|---------------------------------------
PRCOL326I PERFREP .AM_RT_D            |         1         0         1         0
PRCOL326I PERFREP .AM_RT_H            |         1         0         1         0
PRCOL326I PERFREP .AM_RT_M            |         1         0         0         1
PRCOL325I ----------------------------|---------------------------------------
PRCOL327I Total                       |         3         0         2         1
PRCOL003I
PRCOL301I Collect ended at 1998-04-17-03.00.23.
```

5.4 Using the Reporter GUI

Now that everything is set up, you are ready to use the Reporter GUI to create and display reports.

The GUI can be started from the command line, or from the Tivoli Desktop.

5.4.1 From the Command Line

Complete the following steps:

1. Log in as *perfrep*.
2. Export the display to your current location.
3. Issue the command perfrep.

5.4.2 From the Tivoli Desktop

Double click on the Reporter icon on the Tivoli Desktop. This will run the script `/usr/local/Tivoli/bin/aix4-r1/perfrep/bin/perfrep10`.

This script will switch your user ID to *perfrep*, so make sure that the .profile file for *perfrep* exports the display to your current screen location.

Now you should see a screen similar to Figure 50.

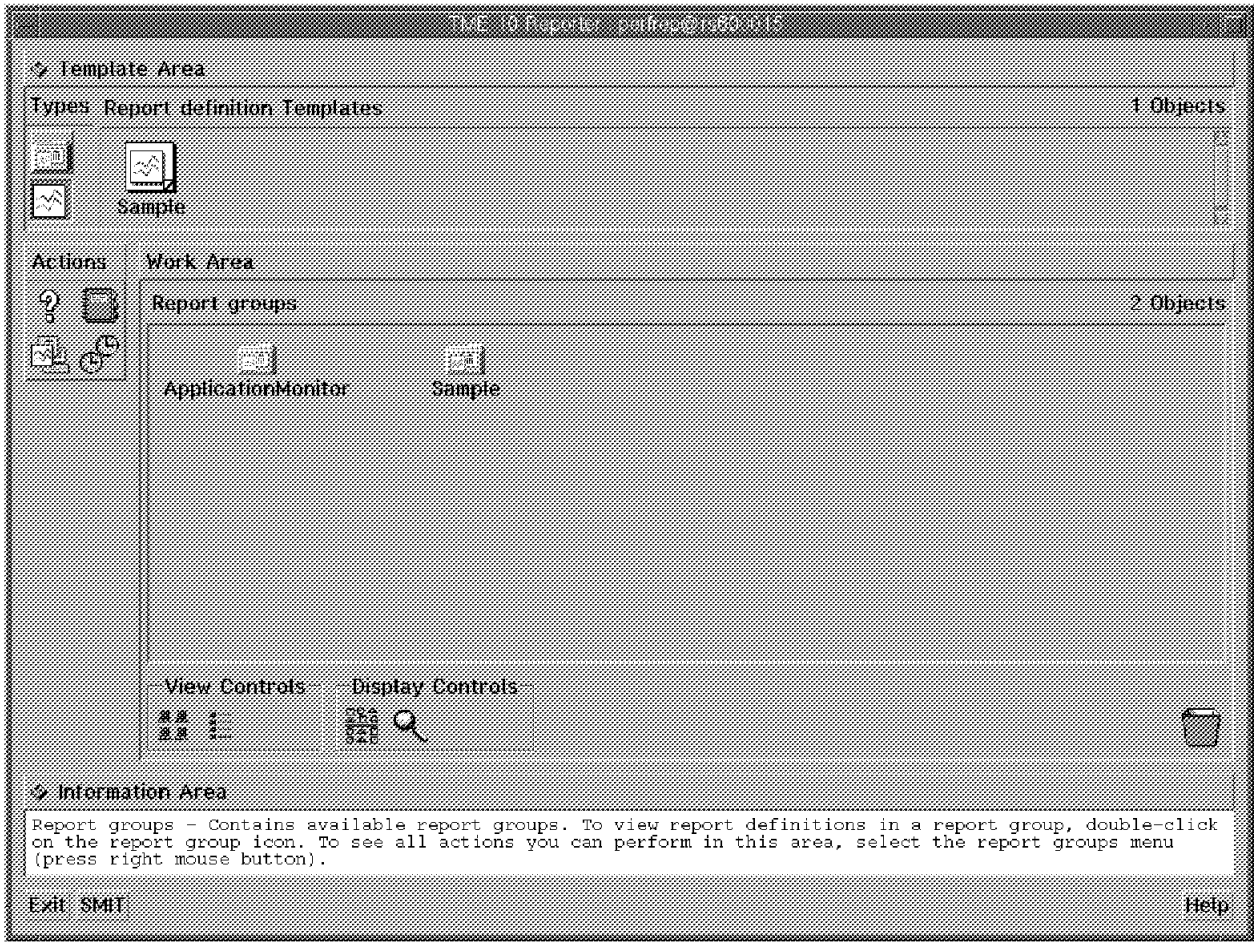


Figure 50. The Reporter Desktop

Any errors will be written into the file `/tmp/perfrep10.out`. This file can be very useful for debugging.

If you are prompted for a database password during Reporter startup, something is wrong. Probably the user ID has not been properly switched to *perfrep*, or the *perfrep* user does not have access to the database.

5.5 Creating Reports from ARM Data

To create reports of ARM data, start by opening the **ApplicationMonitor** icon on the Reporter Desktop.

Now you will see an icon for each type of ARM data report that Reporter can provide, as shown in Figure 51 on page 66.

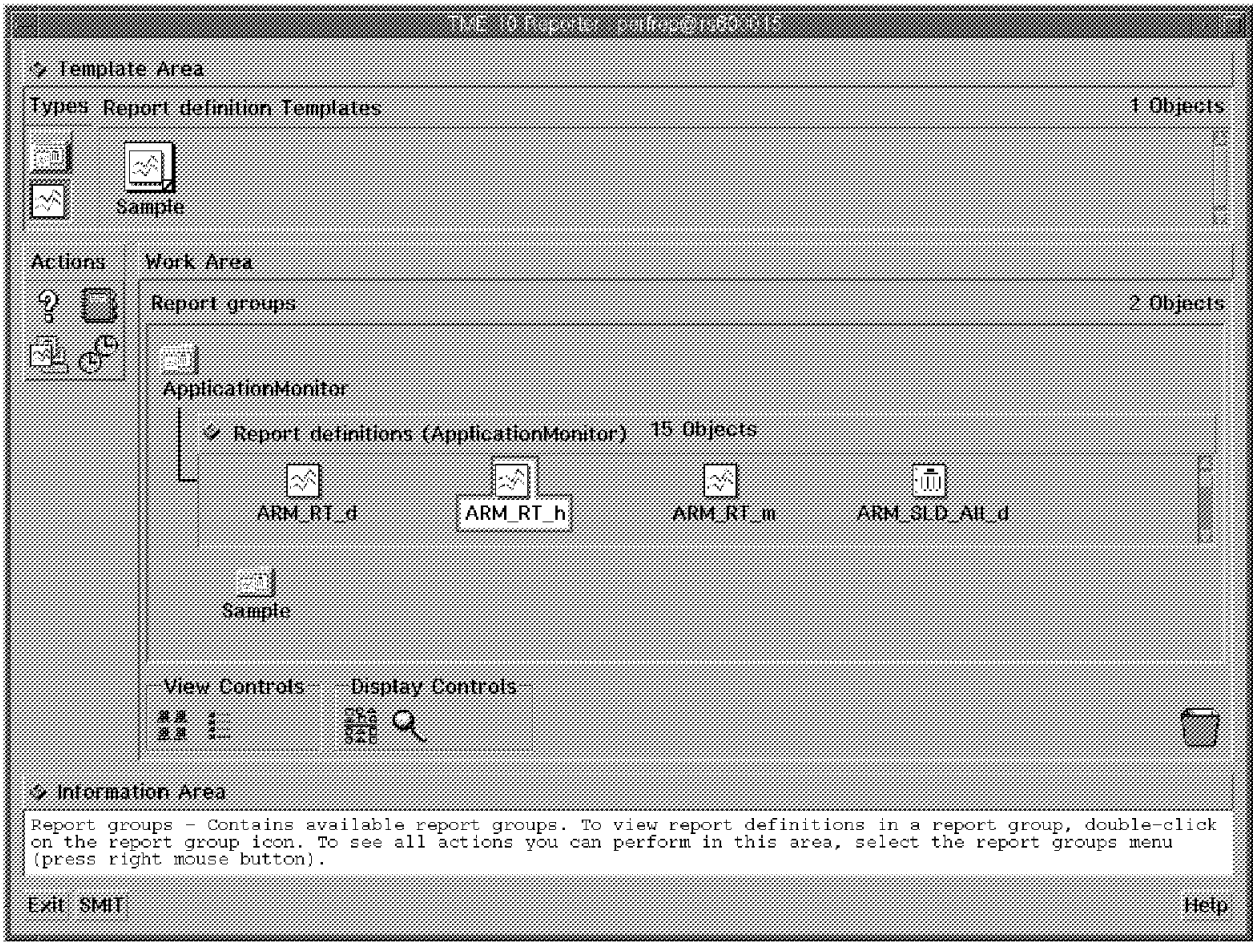


Figure 51. Opening the ARM Reports Interface

If you place the mouse pointer over any one of the icons, a description of the report it represents appears in the Information Area at the bottom of the screen.

There are 15 reports available, as follows:

- Response times, daily overview
- Response times, hourly trend
- Response times, monthly trend
- Service Level details by Application/Transaction/SNMP subagent, daily overview
- Service Level details by Application/Transaction/SNMP subagent, hourly trend
- Service Level details by Application/Transaction/SNMP subagent, monthly trend
- Service Level details by Application/Transaction by node group, daily overview
- Service Level details by Application/Transaction by node group, hourly trend
- Service Level details by Application/Transaction by node group, monthly trend
- Service Level Summary by Application/Transaction/SNMP subagent, daily overview
- Service Level Summary by Application/Transaction/SNMP subagent, hourly trend
- Service Level Summary by Application/Transaction/SNMP subagent, monthly trend
- Service Level Summary by Application/Transaction by node group, daily overview
- Service Level Summary by Application/Transaction by node group, hourly trend
- Service Level Summary by Application/Transaction by node group, monthly trend

We opened the **ARM_RT_h** icon, which represents the *Response times, hourly trend* report.

You can see this report in Figure 52.

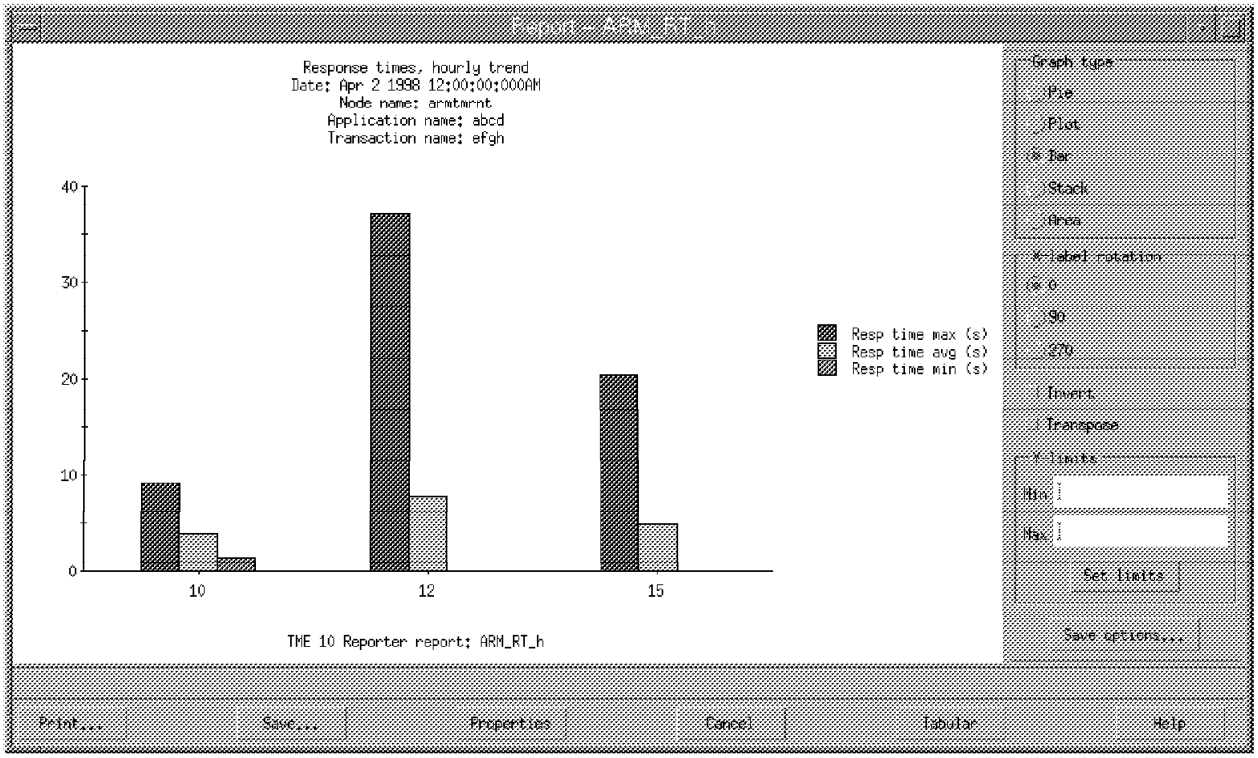


Figure 52. The Hourly Trend Report

We found that, unfortunately, it is not possible to see a report of ARM data for an individual ARM Client Agent machine. The data can be reported by *SNMP subagent*. This means ″by ARM Server Agent″; SNMP is not used by the ARM agents.

Chapter 6. Graphical Monitoring

Distributed Monitoring is primarily an alerting mechanism, but it now has a data capture capability that can log the result of normal monitors to a file.

The contents of this file can be displayed in real time using Java graphics, by any of the popular Web browsers.

This provides a very powerful, and yet easy-to-use interface for ARM data, as you can see in Figure 53.

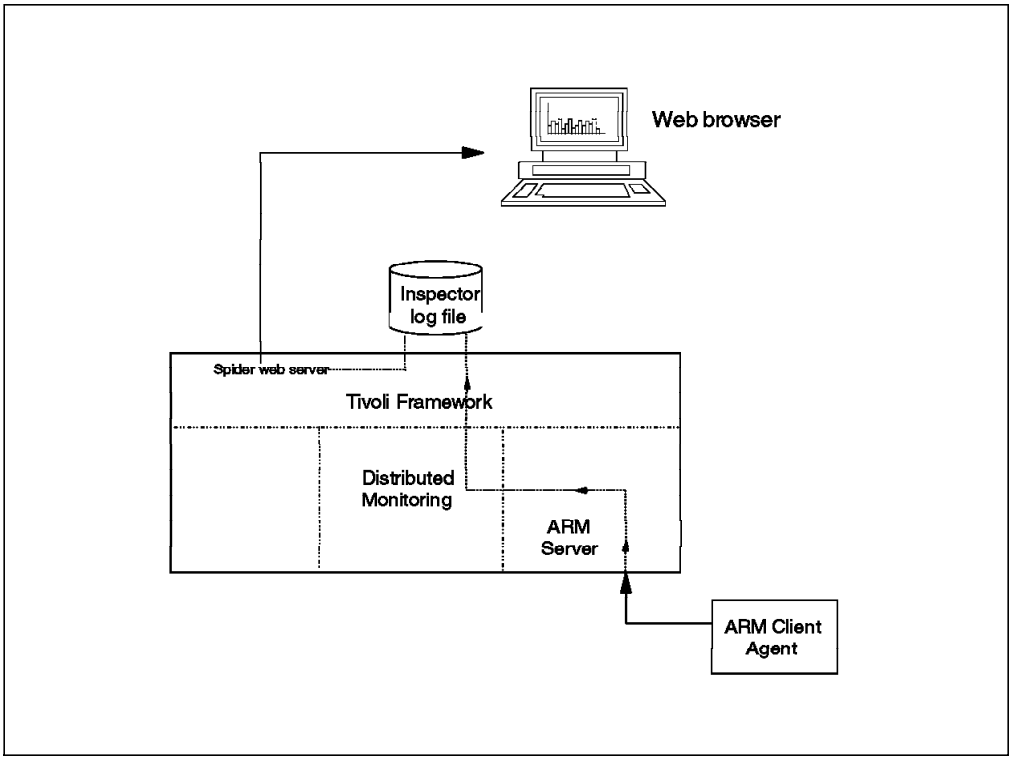


Figure 53. Monitoring ARM Data With a Web Browser

6.1 Using the Graphable Log

The graphical monitoring facility provides a Web-based application for displaying collected results. It includes a specialized Web server, some CGI programs and a set of Java applets. The applets allow you to select the source data and display it on a dynamically updated graph in any Java-enabled browser.

This function is made up of a data capture capability that logs the result of normal monitors to a file and a Web-based monitoring application that displays numerical data in a graphical form, updated dynamically.

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

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

6.1.1 How Spider Works

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

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

Figure 54 illustrates how Spider is related to oserv and how a user gains access to it.

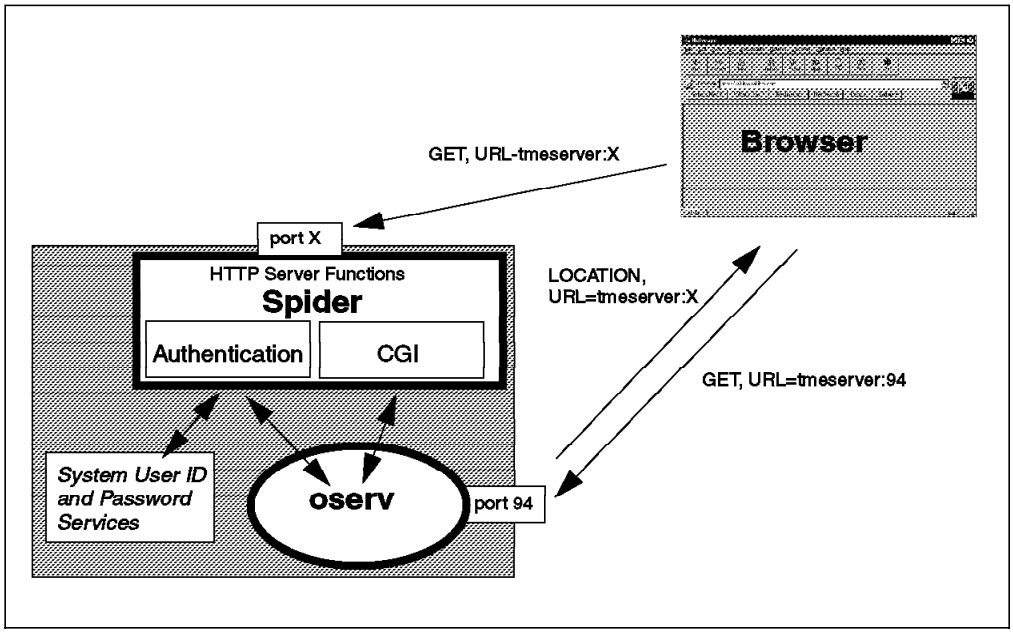


Figure 54. Spider Web Server Connections

You just need to define some Sentry monitors to collect data, so that you can display it in graphical form with Spider.

6.2 Creating a Distributed Monitoring Profile

A Distributed Monitoring profile is required if you want to display data graphically, or if you want to set up asynchronous monitors.

6.2.1 Creating A Profile Manager

Open the ARM Policy Region from the TME Desktop, then click on **Create** on the toolbar, and select **Profile Manager** on the pull-down menu that appears. You should now see a screen similar to Figure 55.

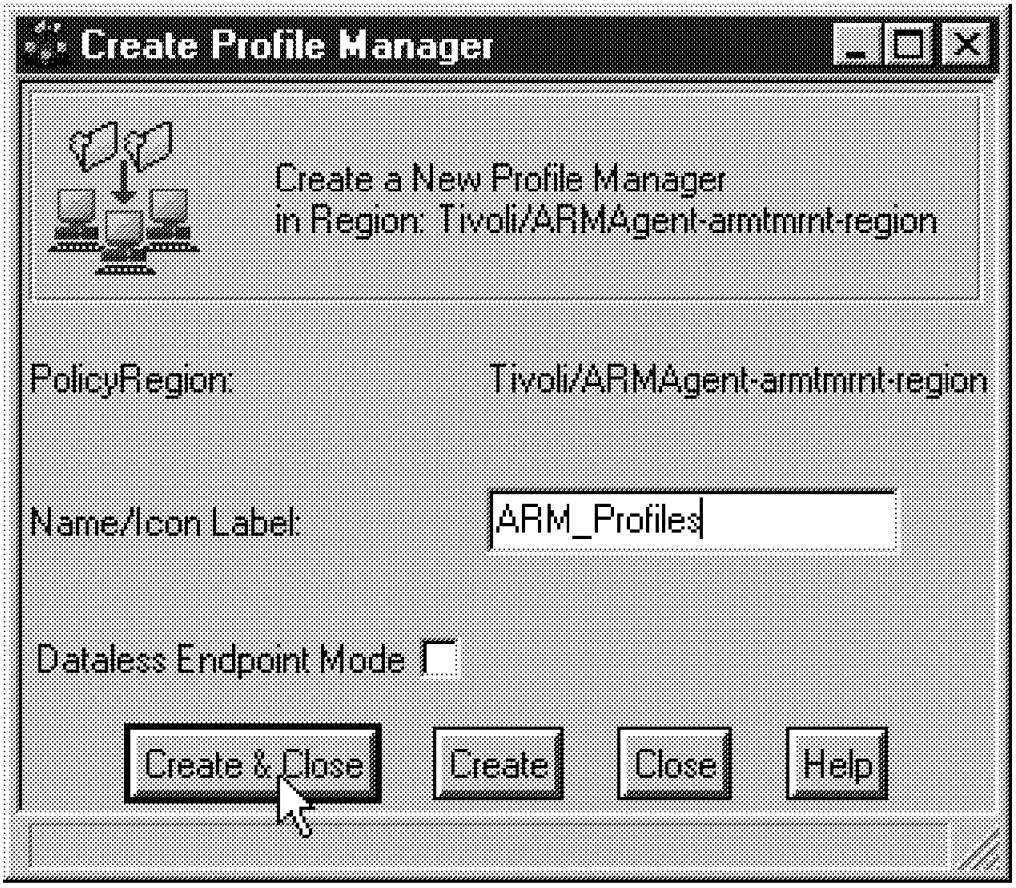


Figure 55. Creating a New Profile Manager

Type in a name for your Profile Manager. We recommend using a meaningful name that will be easily identifiable on the Desktop; we used *ARM_Profiles.*

Now click on **Create & Close**.

6.2.2 Creating A Profile

Open the newly created **Profile Manager** icon, then click **Create** on the toolbar, and select **Profile...** from the pull-down menu that appears. Now you should have a screen similar to Figure 56 on page 72.

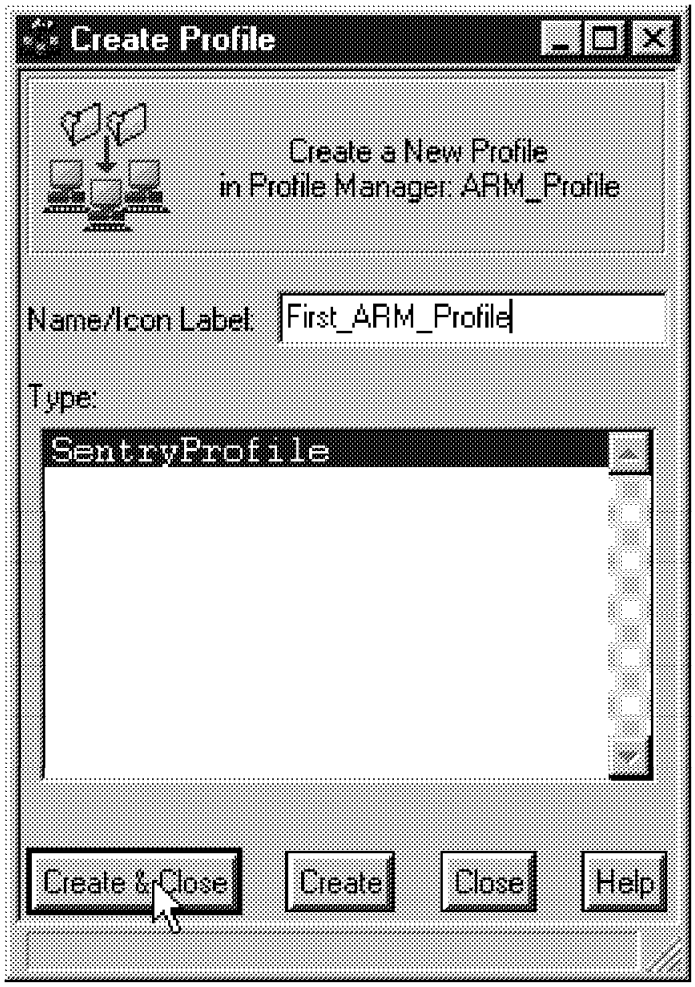


Figure 56. Creating a New Profile Within the Profile Manager

Type in a profile name, and click **Create & Close**.

Distributed Monitoring versus Sentry

The naming standard used here is a little confusing. Distributed Monitoring is the new name for Sentry; you are actually creating a Distributed Monitoring Profile.

Figure 57 on page 73 shows the icon which was created in the Profile Manager for our new profile.

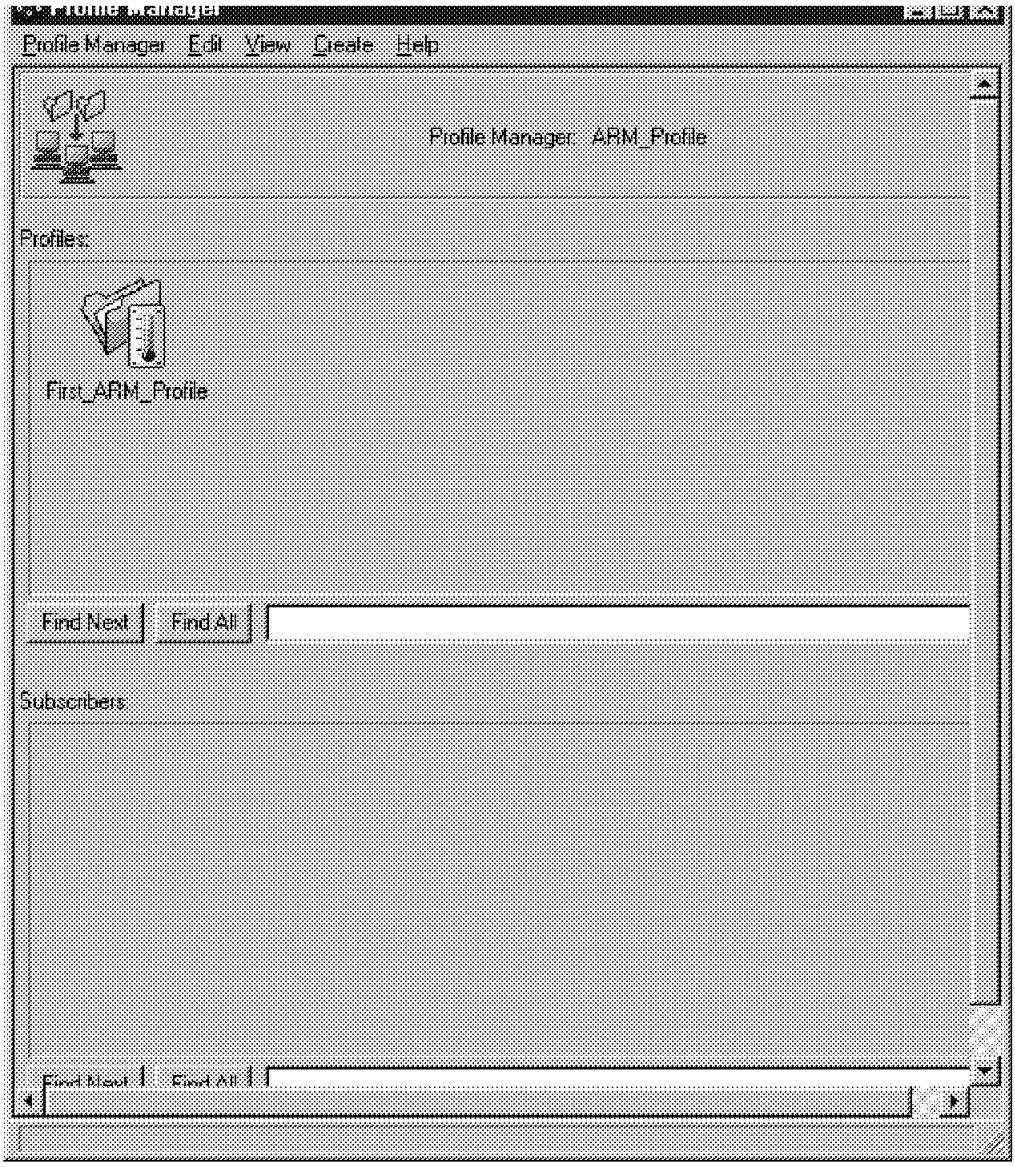


Figure 57. The Newly Created Profile

6.2.3 Adding Monitors to the Profile

Complete the following steps to add monitors to the profile:

1. Open the profile by double-clicking on its icon.
2. Click on **Add Monitor**, as shown in Figure 58 on page 74.

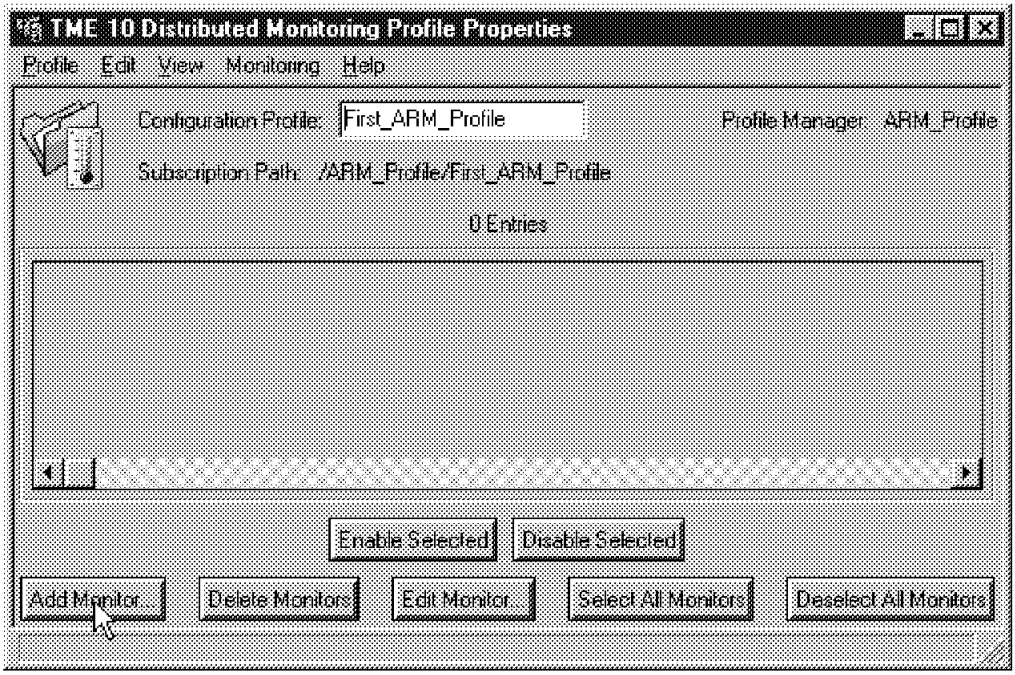


Figure 58. Selecting to Add Monitors to the Profile

3. In the Add Monitor dialog box, shown in Figure 59 on page 75, select **ARM** from the list of Monitoring Collections, and the available ARM metrics will appear in the Monitoring Sources pane.

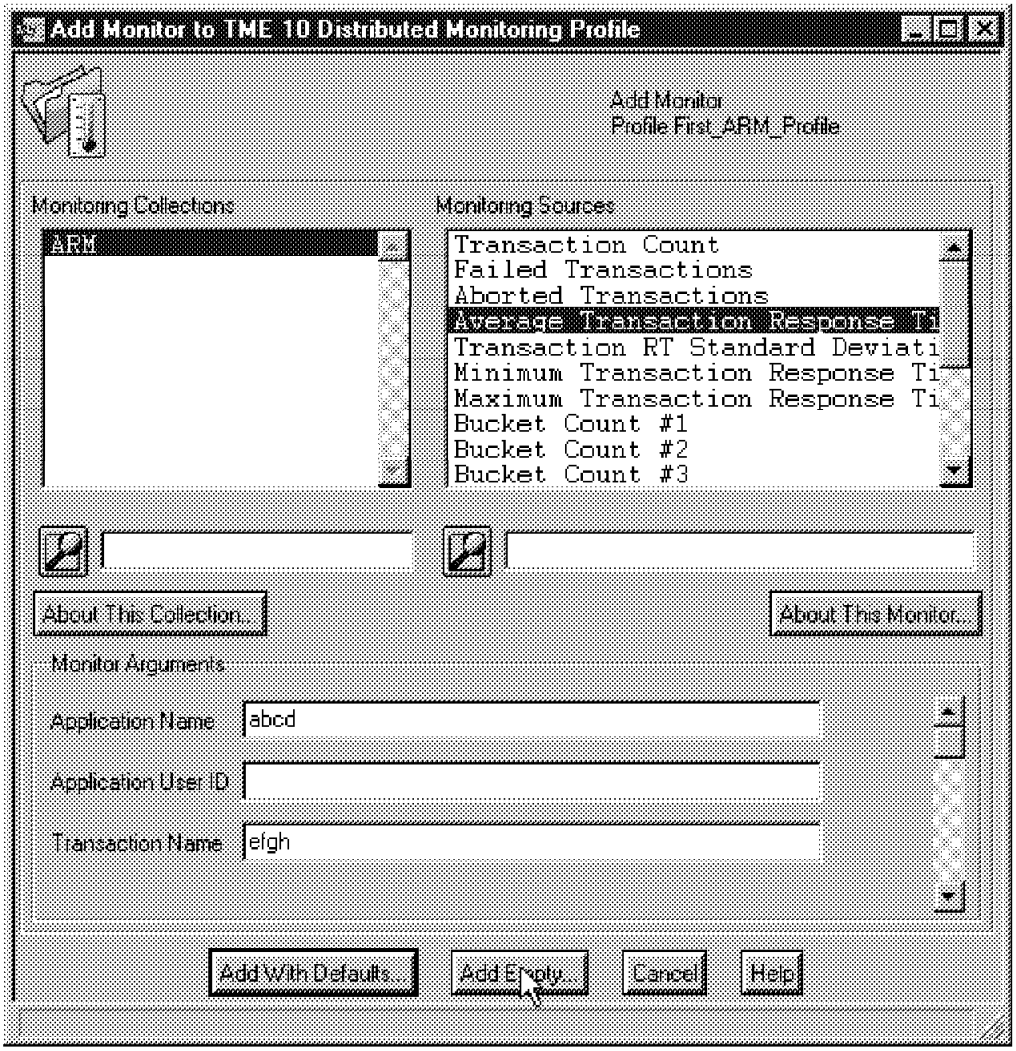


Figure 59. The Add Monitor Dialog Box

4. Select the metric that you want to collect. We chose **Average Transaction Response Time**.

   For this, and most of the other metrics, you will need to specify monitor arguments.

5. So if required, specify the Application Name and Transaction Name.

   Leave the Application User ID field blank if you want to collect data for all users of this application/transaction pair, or type a specific ID.

6. When you are done, click on **Add Empty**.

7. Now you will be presented with a screen similar to that shown in Figure 60 on page 76.

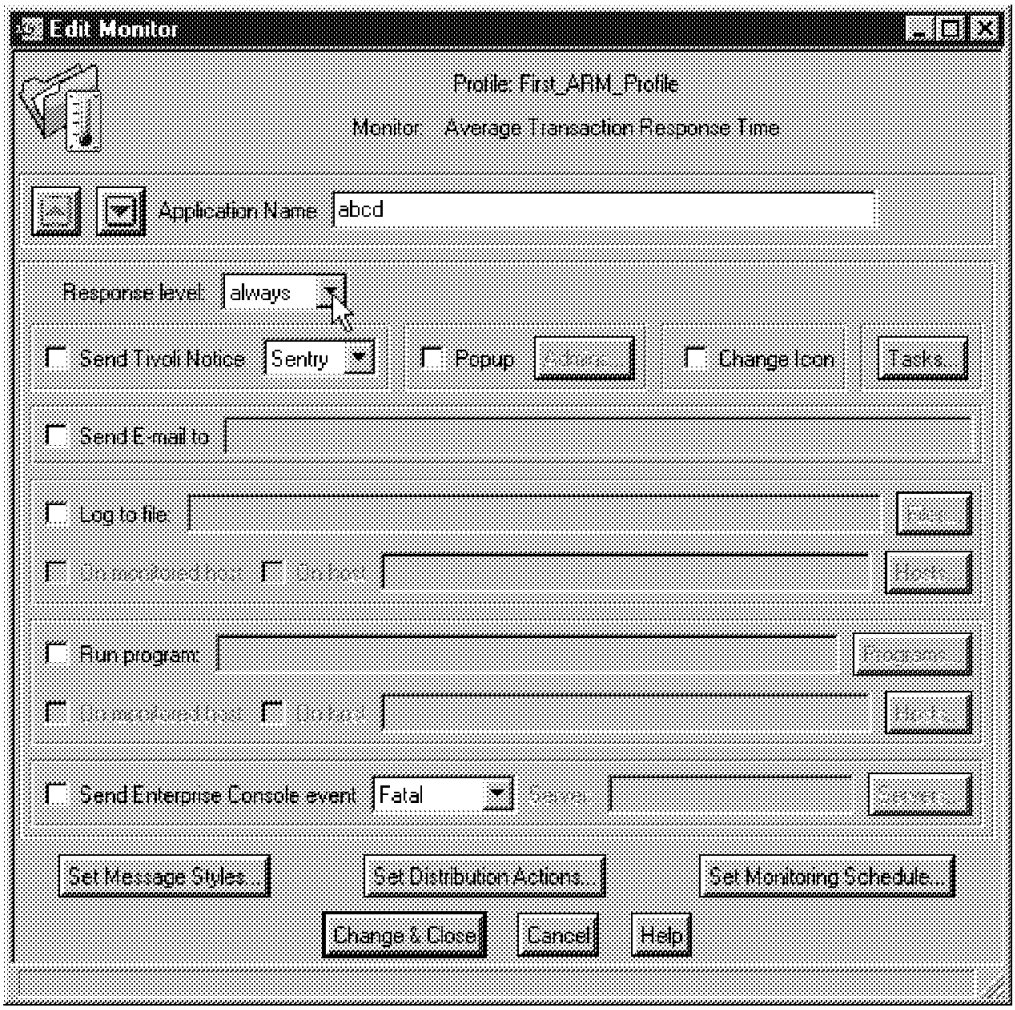


Figure 60. The Edit Monitor Dialog Box

We want Distributed Monitoring to write into the graphical log each time the monitor is invoked, so select a response level of **always**.

Now click on the **Tasks** button to open the Response Task dialog box.

In the dialog box, double-click **Sentry Graphable Logs** as the task library and then double-click on the **Create graphable log** task, as shown in Figure 61 on page 77.

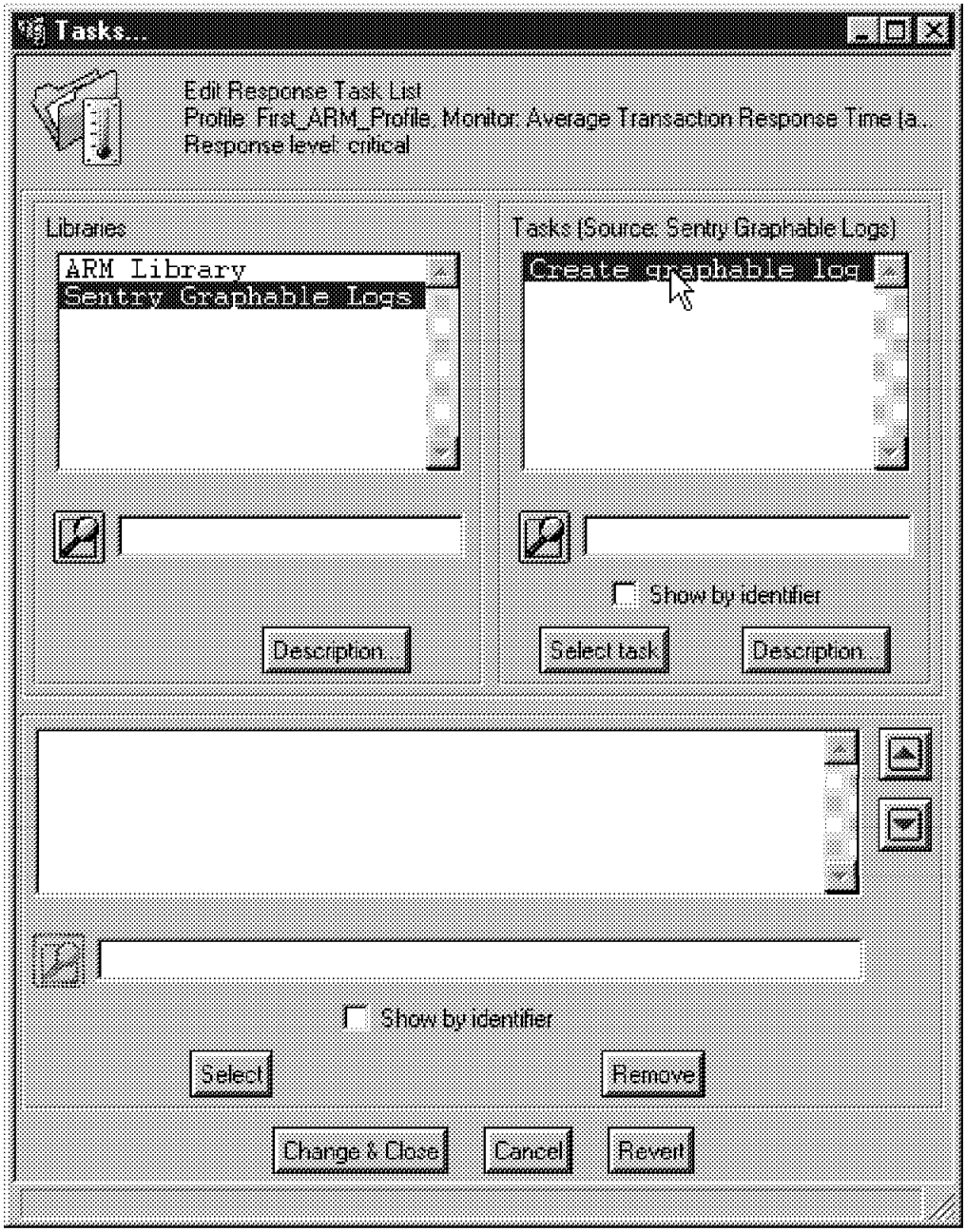


Figure 61. The Response Task Dialog Box

Now you will be prompted to specify how many lines of data should be kept in the graphable log. See 6.3, “The Graphable Log File” on page 83 for a discussion of log file size. The default is 1000, as you can see in Figure 62 on page 78.

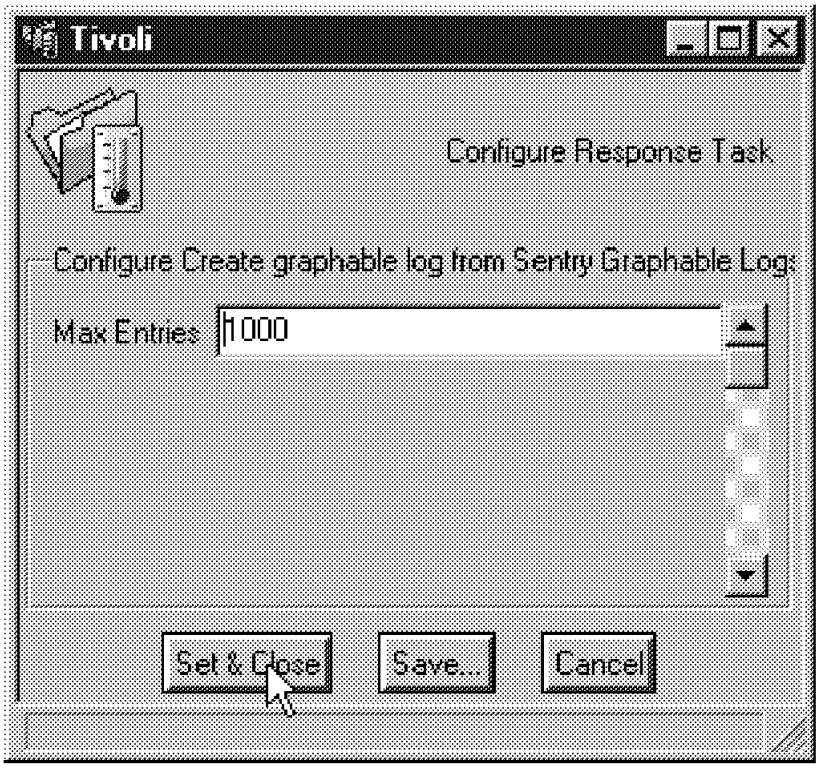


Figure 62. Setting the Number of Entries in the Graphable Log

Click on **Set and Close** to return to the Response Task dialog box.

Now `Create Graphable Log(1000)` should appear in the lower pane, as shown in Figure 63 on page 79.

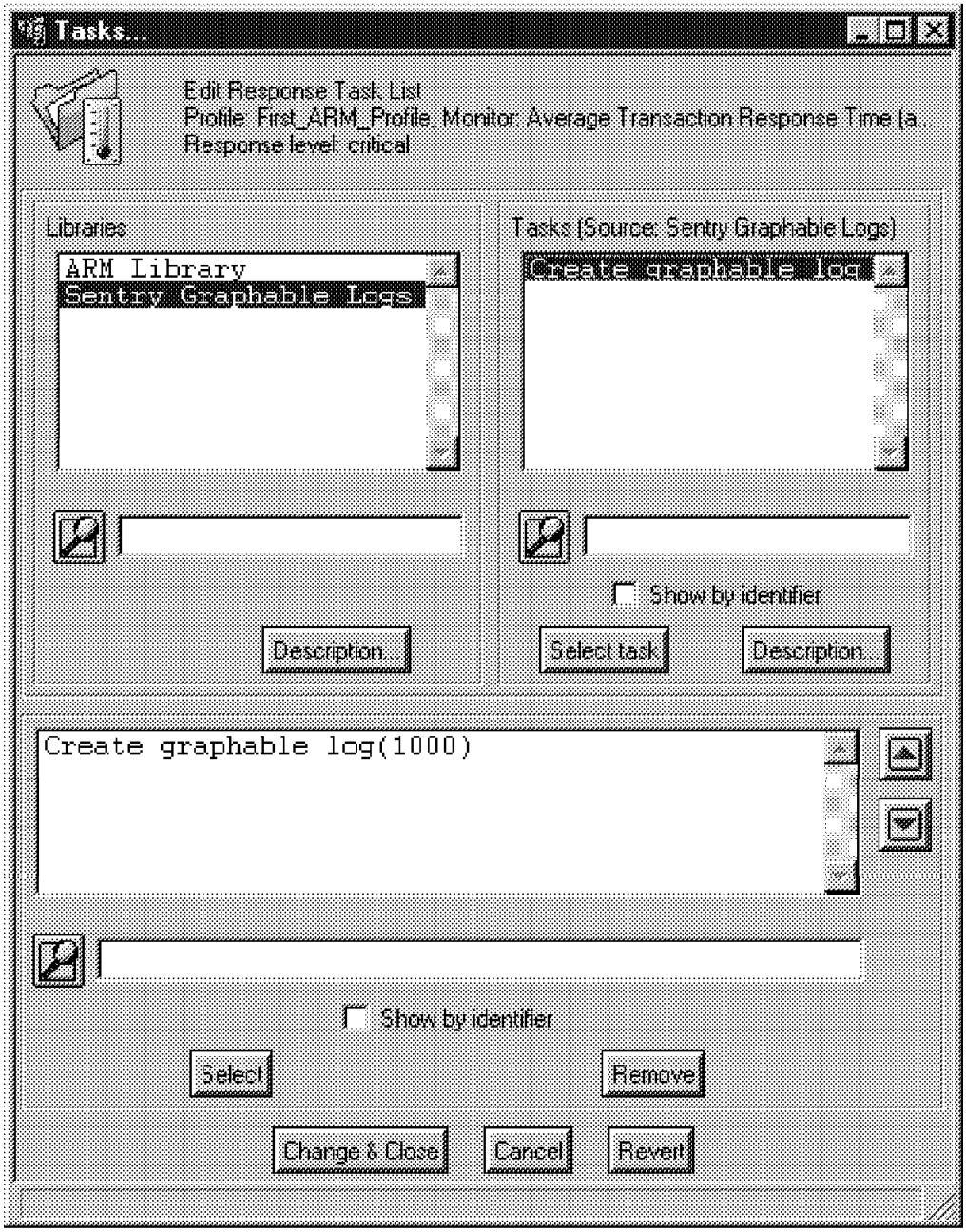


Figure 63. The Log Size is Set

Click on **Change and Close** to return to the Edit Monitor dialog box.

Now click on **Set Monitoring Schedule...** to set the monitoring frequency. The default is hourly.

Click on **Change & Close** to return to the Edit Monitor dialog box.

Repeat the above steps if you want to collect more than one metric.

Now click on **Change & Close** to return to the Profile Properties dialog box.

After the monitors are created, there is a separate **Save** step that must be performed from the Profile pull-down in the Profile Properties screen as shown in Figure 64 on page 80.

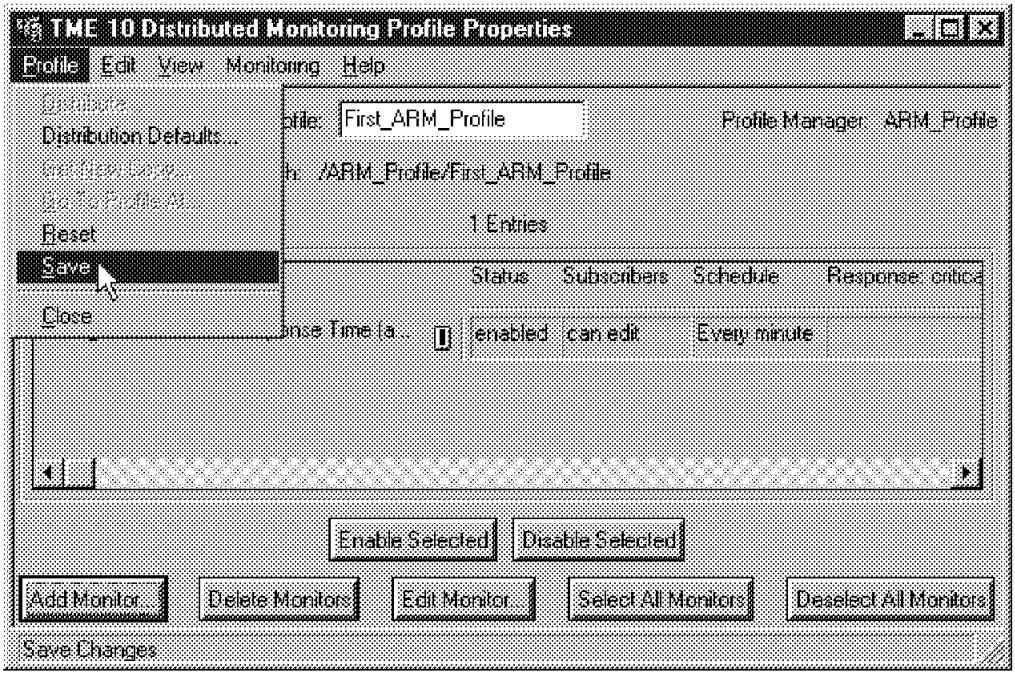


Figure 64. Saving the Profile

Close this dialog box, to return to the Profile Manager dialog box.

6.2.4 Selecting Subscribers to the Profile

Before you can distribute the profile, you must first select subscribers to it.

Select the profile in the Profile Manager dialog box, and from the Profile Manager pull-down, select **Subscribers...**.

Now choose the subscribers from the Available to become Subscribers list. Note that Distributed Monitoring profiles can only be distributed to proxies, so the only valid subscribers are proxies.

We made our Distributed Monitoring Proxy a subscriber, as you can see in Figure 65 on page 81.

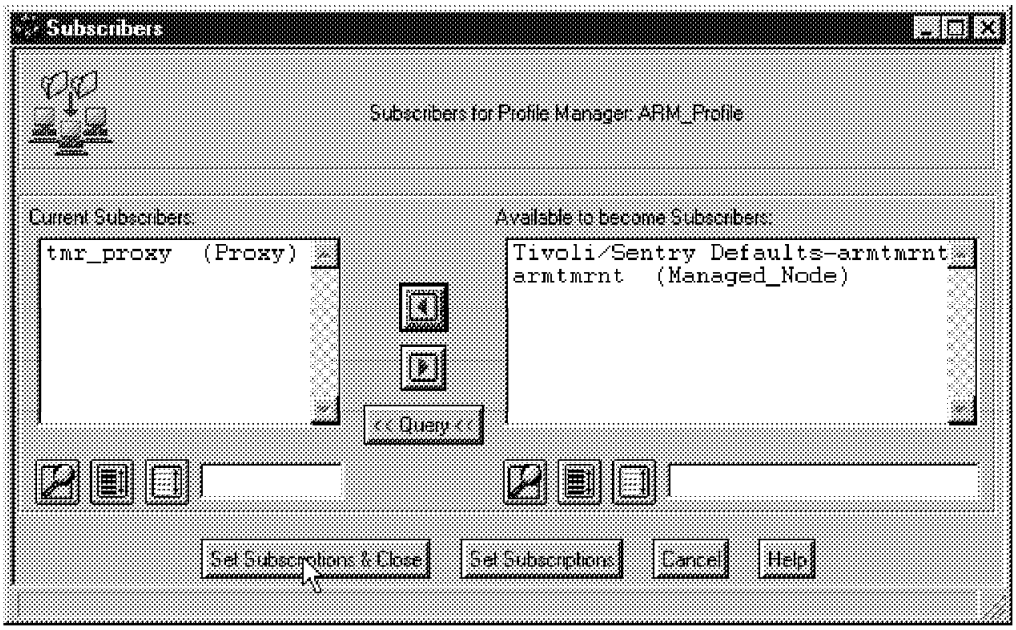


Figure 65. Selecting Subscribers to the Profile

After making your selections, click on **Set Subscriptions & Close**.

6.2.5 Distributing the Profile

The profile should be distributed to the proxy endpoints immediately after the ARM Server Agent is started.

To distribute the profile, in the Profile Manager dialog box click with the left mouse button on the profile to be distributed and (again with the left mouse button) on the subscriber to which you want to distribute.

Next, click on **Profile Manager** on the toolbar, and select **Distribute** from the pull-down menu, as shown in Figure 66 on page 82.

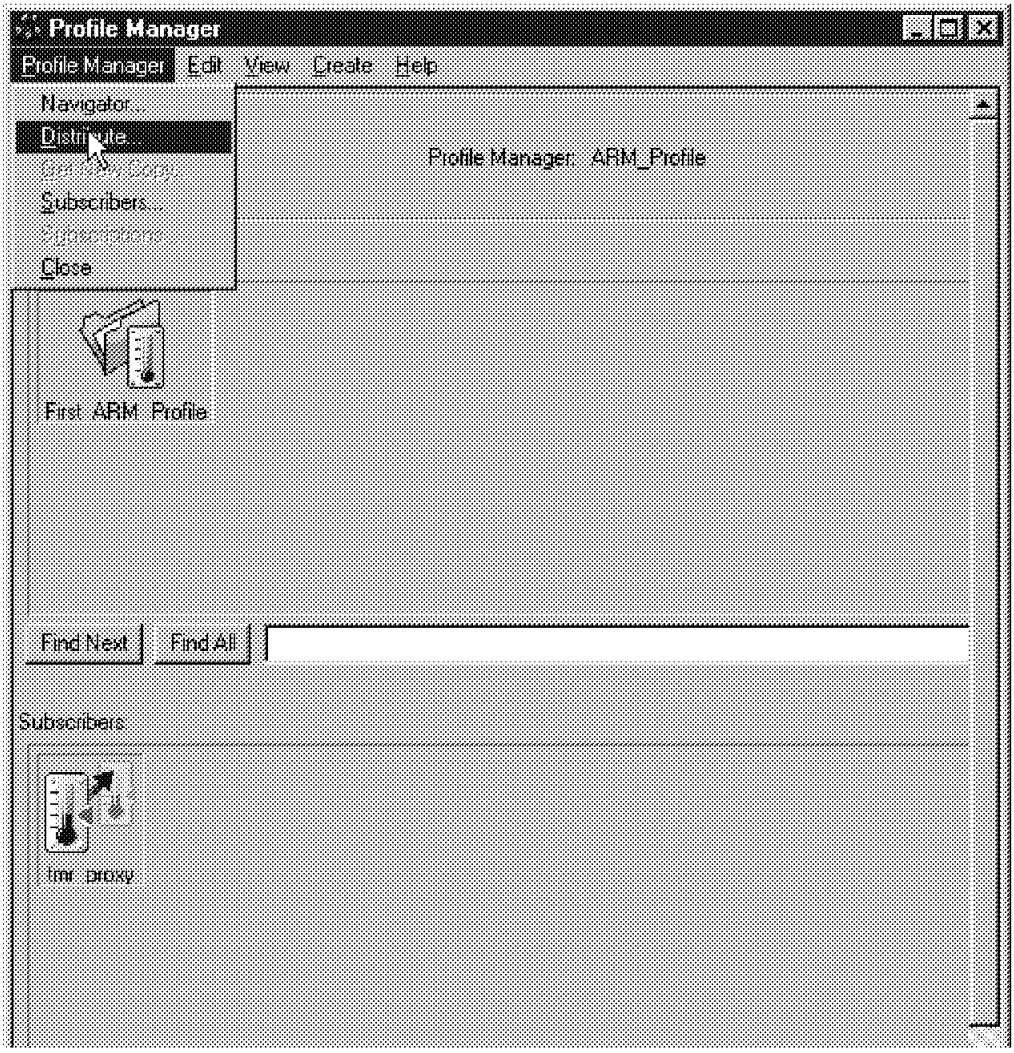


Figure 66. Distributing the Profile

Now select either **Distribute Now**, or **Schedule**, to distribute the profile at a later time.

On the Tivoli Desktop, you should see messages similar to this in the Operation Status panel:

```
Distributing profile First_ARM_Profile...
Distributed profile First_ARM_Profile.
```

To check that they were distributed correctly, double-click on the subscriber, and in the resulting window, double click on the profile. You should see the contents of the profile that is now on the subscriber.

The ARM Client Agent needs to be restarted after a new or changed profile is distributed to its proxy endpoint. As soon as this is done, monitoring will start with the new profile.

6.3 The Graphable Log File

At the end of each monitoring interval, a data point will be added to the graphable log file.

Each data point in the file consists of an 8-digit time stamp, the monitor value and the text of the monitor status (critical, severe, warning, or normal).

The graphical data is stored on the managed node where the Sentry engine is running, under the Tivoli database directory, and has a path and name similar to:

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

Where:

- $DBDIR is the Tivoli database directory on the managed node.
- <Systemname> is the name of the system where the data was collected.
- <Profilename> is the name of the Distributed Monitoring profile containing the monitor.
- <Monitor_type> is a string identifying the type of monitor.
- <ObjID> is an identifier containing an oserv object ID number.

In our case, we had, for example:

```
Tivolidbarmtmrnt.db.sntglogtmr_proxy
First_ARM_Profile\ARM_AvgTxRT_0e1483757716.1.609
```

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

6.4 Viewing ARM Data from the Graphable Log

In order to see the ARM data using a Web browser, you must make sure that the Spider process is running on the machine where the ARM server is installed.

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

```
wstarthttpd
```

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

http:*hostname*:94

Where *hostname* is the name of the managed node where the ARM server and Spider daemon are running. In our case this was *armtmrnt*.

Port 94 will find the dynamic port chosen for the Spider Web server and automatically point you to the correct URL.

In the initial page click on the hypertext link to **TME 10 Distributed Monitoring Inspector**, as shown in Figure 67 on page 84.

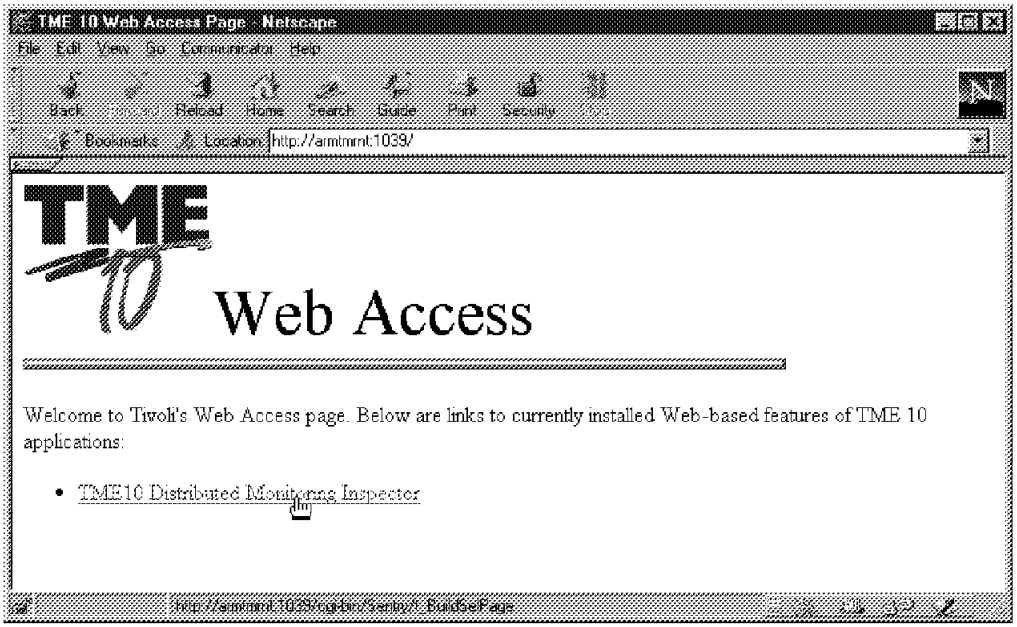


Figure 67. Accessing the Inspector

Now you will be prompted for a user ID and password.

You must supply a system user ID on the TMR server. There are two things you should consider here:

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

Enter the ID and password and click on **OK**.

Now you will be presented with a screen similar to Figure 68 on page 85.

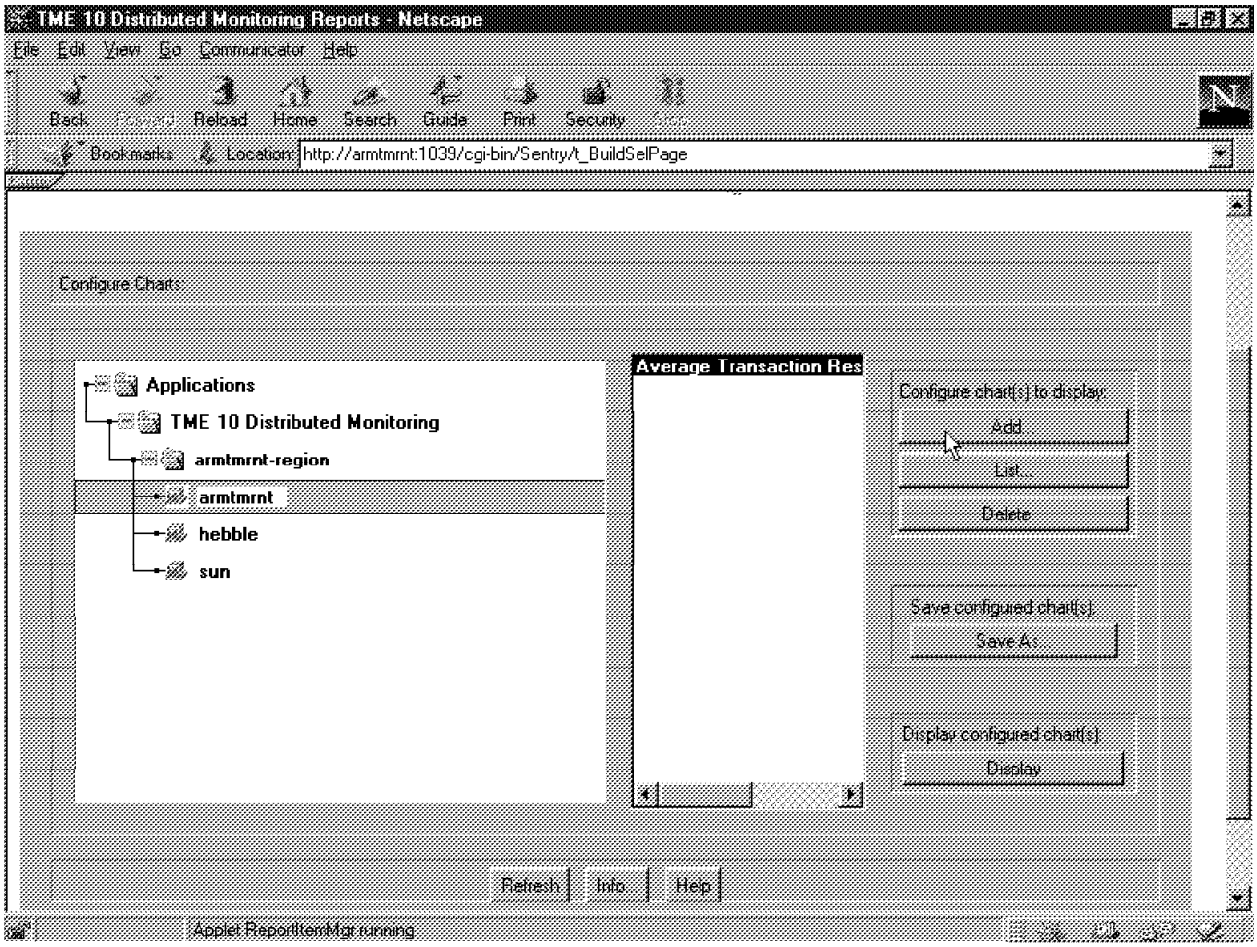


Figure 68. The Browser View of ARM Resources

Initially you will only see the Applications option. Open up the resource tree by clicking on the plus sign next to each icon.

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

6.4.1 Defining the Reports

Now you can add graphical reports. You can have combinations of multiple monitors and multiple nodes. To create a graph:

1. Select the combination of nodes and monitors that you want to see. We selected our TMR server node and the Average Transaction Response Time monitor.
2. Click on **Add**, to configure a chart.

The screen shown in Figure 69 on page 86 will pop up. Select the options you want and click on **Add/Close**.

Figure 69. Report Options Screen

You will return to the main report selection screen.

When you have created the graph that you want, click on **Display**. A new browser window will start up, containing a number of Java applets.

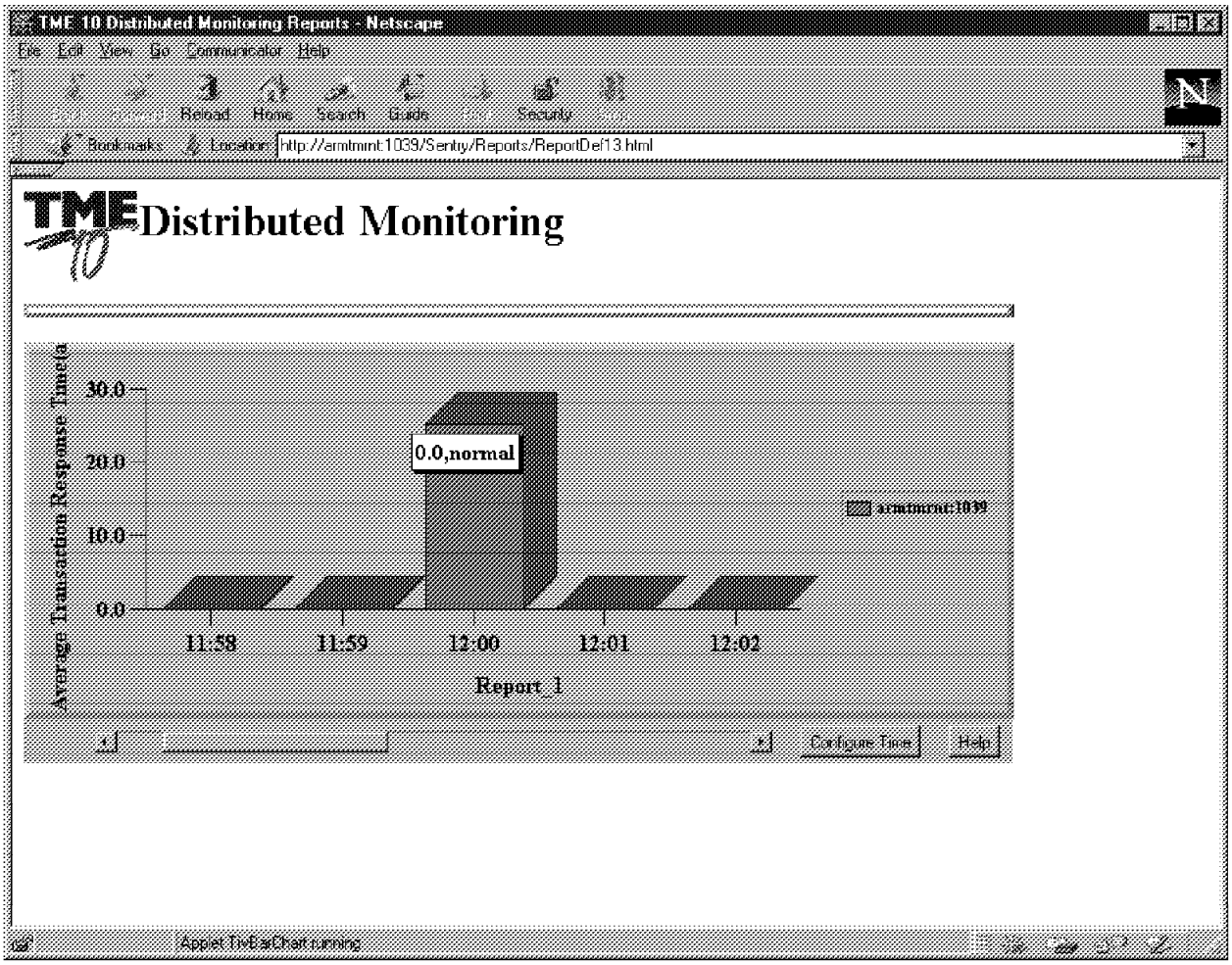


Figure 70. Graphical Reports in Action

You can use the scroll bar to look at data collected prior to the range of times shown on the x-axis. By default the graphs show the most current figures and update automatically when new data is logged. If you stop the mouse pointer over a data point, the value of it pops up in a little box. Unfortunately, in the release of code that we had, the pop-up did not always display the correct values.

Chapter 7. Production Use of the ARM Agents

In this chapter we cover some of the things that you should consider when putting the ARM agents into production use.

7.1 Server to Client Traffic

When collection is running on an ARM Client Agent machine, the amount of data that is sent to the ARM Server Agent is small, and it is sent only at the end of the collection interval; it is not sent after each transaction completes.

However, if you start collection with a 60-second interval, data is sent from the client to the server when the system clock on the client is at 00. This could have serious implications if you synchronize the system clocks of all your client machines; they will all send to the server at the same moment, creating a traffic blip.

7.2 Setting Thresholds in the Agents and Asynchronous Monitoring

Once you have your ARM environment installed and working, you may want to use some of the capabilities of Distributed Monitoring to warn you when application performance is worse than it should be.

You can define asynchronous monitors for ARM, so that when the conditions of the monitors are met, a task can be run to notify you, or maybe even fix the problem.

There are three types of asynchronous monitors for ARM:

TxHung The transaction did not update or stop within the threshold specified.

TxExceedMaxLen The transaction did not stop within the time specified.

TxStatus The transaction failed or aborted.

To set up asynchronous monitoring, you will need to complete the following tasks:

1. Define the monitors and their thresholds in an ARM client configuration file
2. Define the actions that should occur when the monitor triggers
3. Activate monitoring

7.2.1 Defining Monitors in a Client Configuration File

The configuration file is optional for the ARM Client Agent, so you will need to create it, and then fill in the monitor definitions in the following format:

```
SERVERNAME= hostname

MONITOR
  APPLICATION=appl_name
  USERID=userid
  TRANSACTION= tran_name
  TYPE= monitor_type
  VALUE = time_in_miliseconds | failed |aborted.
```

You can define as many monitors as you want. The parameters in the definition for each monitor are as follows:

APPLICATION The name of the application that you want to monitor.

USERID The user of this application that you want to monitor. An asterisk can be specified, so that all users of the application are monitored.

TRANSACTION The name of the transaction that you want to monitor

TYPE The type of monitor that you want. This can have the following values:

- TxHung
- TxExceedsMaxLen
- TxStatus

VALUE If the monitor type is *TxHung* or *TxExceedsMaxLen*, then this value must be the time_in_milliseconds of the threshold. If the monitor type is *TxStatus*, then it can be *FAILED* to indicate transactions that failed, or *ABORTED* to indicate those that aborted.

Failed and Aborted

The completion status of the transaction is set by the *arm_stop* API call, and can be *GOOD*, *FAILED* or *ABORTED*.

See the *Application Response Measurement API Guide*, second edition, for more details on how this status field can be used.

We set up a monitor to trigger when the *efgh* transaction in the *abcd* application runs for more than five seconds. You can see our configuration file below:

```
SERVERNAME=armtmrnt

MONITOR
  APPLICATION=abcd
  USERID= *
  TRANSACTION= efgh
  TYPE=TxExceedsMaxLen
  VALUE= 5000
```

7.2.2 Defining the Actions That Occur When the Monitor Triggers

Now we need to set up the monitor and its actions in a Distributed Monitoring profile.

We had created a Profile Manager that contained a profile with a set of subscribers.

Then we defined an asynchronous monitor, following these steps:

1. Open the profile manager, and open the profile.
2. In the TME 10 Distributed Monitoring Profile Properties window, click on **Add Monitor**, and in the Monitoring sources section of the window, choose one of the following:
   - Hung Transaction
   - Transaction too long
   - Failed transaction
   - Aborted transaction
3. Click on **Add Empty**.
4. In the window that appears for the monitor, perform the following selections:
   - Set the Response Level to *always*, so that when the monitor triggers there will always be some action.
   - Select one or more actions that should occur when the monitor triggers. The available actions are as follows:
     - Pop up a window.
     - Send e-mail.
     - Run a specific task.
     - Log the monitor to a file.
     - Send Tivoli Notice.
     - Run a program.
     - Send Enterprise Console Event.
   - Click on **Change & Close**.
   - Once the monitor has been defined, click on **Profile** on the taskbar, and then click on **Save**.
5. Distribute the changed profile to the subscribers.

7.3 Activating the Monitors

Once all the definitions have been made, the ARM Server Agent must be started with the Forward asynchronous event to DM option enabled. On the command line, type:

```
warmcmd srvstart -S
```

If the ARM Server Agent is already running, you can enable event forwarding with the command:

```
warmcmd dmenable
```

Now start the ARM Client Agent with the monitors defined in the configuration file.

To start the client from the command line, type:

```
warmcmd clntstart -f \client.cfg
```

where `\client.cfg` is the path and file name of the ARM Client Agent configuration file.

It is not necessary to start a collection.

7.4 Server to Client Ratio

It is very difficult for us to make a recommendation on the number of ARM Client Agents that you can reasonably connect to one ARM Server Agent. The factors to consider in determining this are as follows:

- The load of other tasks running on the ARM Server Agent machine.
- Collection interval for collections active in the ARM Client Agent.
- Number of collections active on the ARM Client Agent.
- Number of ARM-instrumented applications actually in use on the ARM Client Agent machines at any given time.

7.5 Automating the Transfer of Log Data to Reporter

Once you have set up the task to transfer the ARM log file to Reporter, we suggest that you automate this function so that it runs every day, which is what Reporter expects.

The Tivoli scheduling service can schedule a job, but cannot schedule a task, so in order to automate the transfer task, you will need to complete the following steps:

1. Create a job that runs the task
2. Schedule the job

7.5.1 Creating a Job

To create a job that can be executed from the ARM library or set up to run as a scheduled job, complete the following steps:

1. In the ARM Task Library window, click on **Create** and select **Job** from the pull-down menu, as shown in Figure 71 on page 93.

Figure 71. Starting to Create a Job

2. In the Create a new Job panel, give the job a name. This will become the name of the icon representing the job in the Task Library.
3. Select the task that you want the job to execute, and the type of execution that you desire. We let it default to parallel.

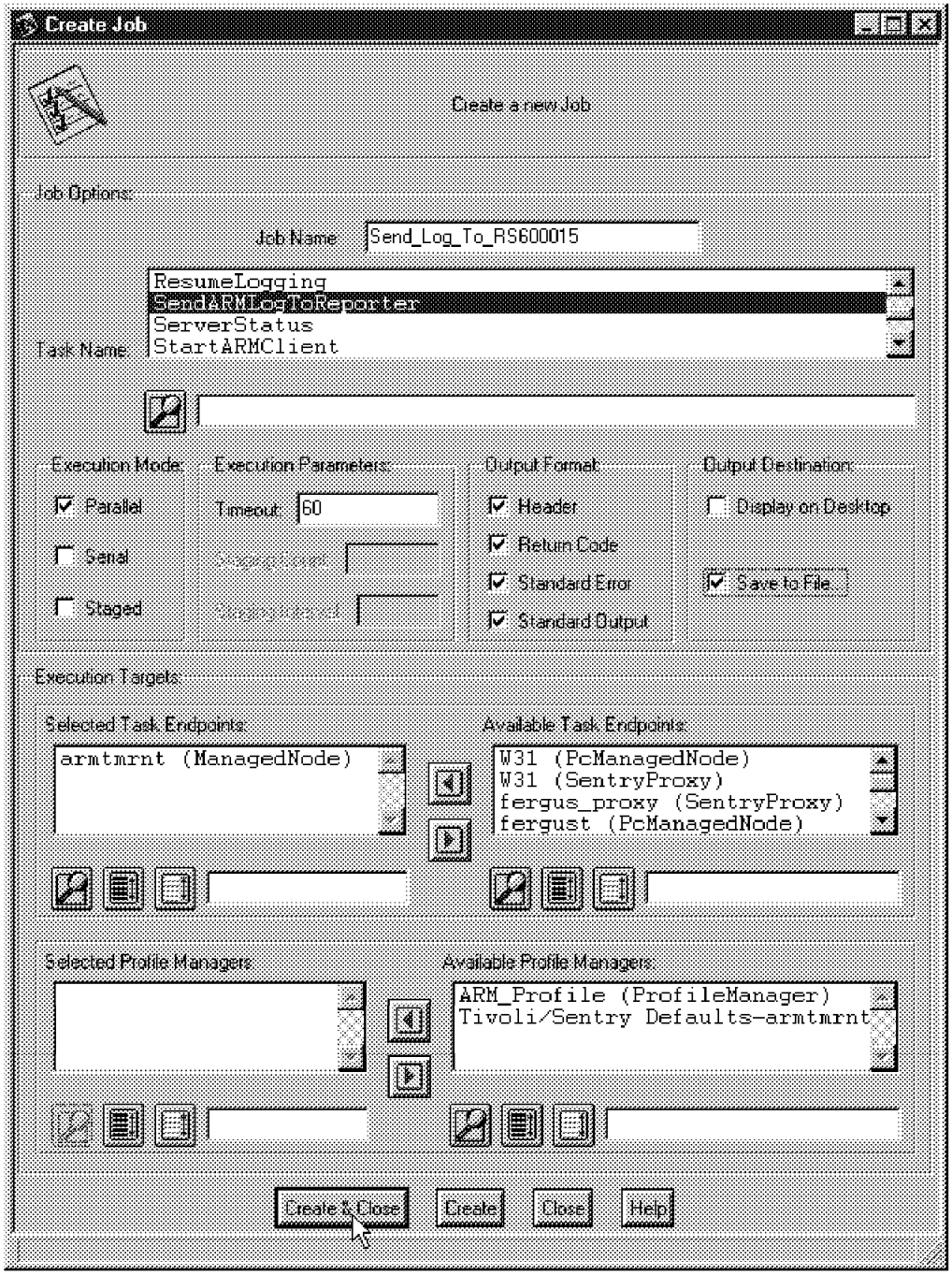


Figure 72. Setting up the Job

4. We chose to save the output of this task to a file, so we were prompted to specify the file name, and the hostname of the machine where it would be created.
5. Click on **Create & Close**, and a new icon will appear in the Task Library.

7.5.2 Scheduling the Job

Now that you have created the job, it can be scheduled, so that it runs automatically whenever you want.

To create a scheduled job the following steps will need to be completed.

1. Click on the icon for the job, and drag it to the Scheduler icon on the Tivoli Desktop.
2. The task arguments dialog box pops up. After completing this with the Reporter login name and the ARM log file name, click on **Set & Execute**.
3. Now the Schedule Job panel pops up.

   In the Job Label box you must enter a label for the job; the scheduler uses this to identify the job. We chose **SLT_RS600015**, as you can see in Figure 73 on page 96.

   In the Schedule Job For field, enter the date and time you wish to have this job run.

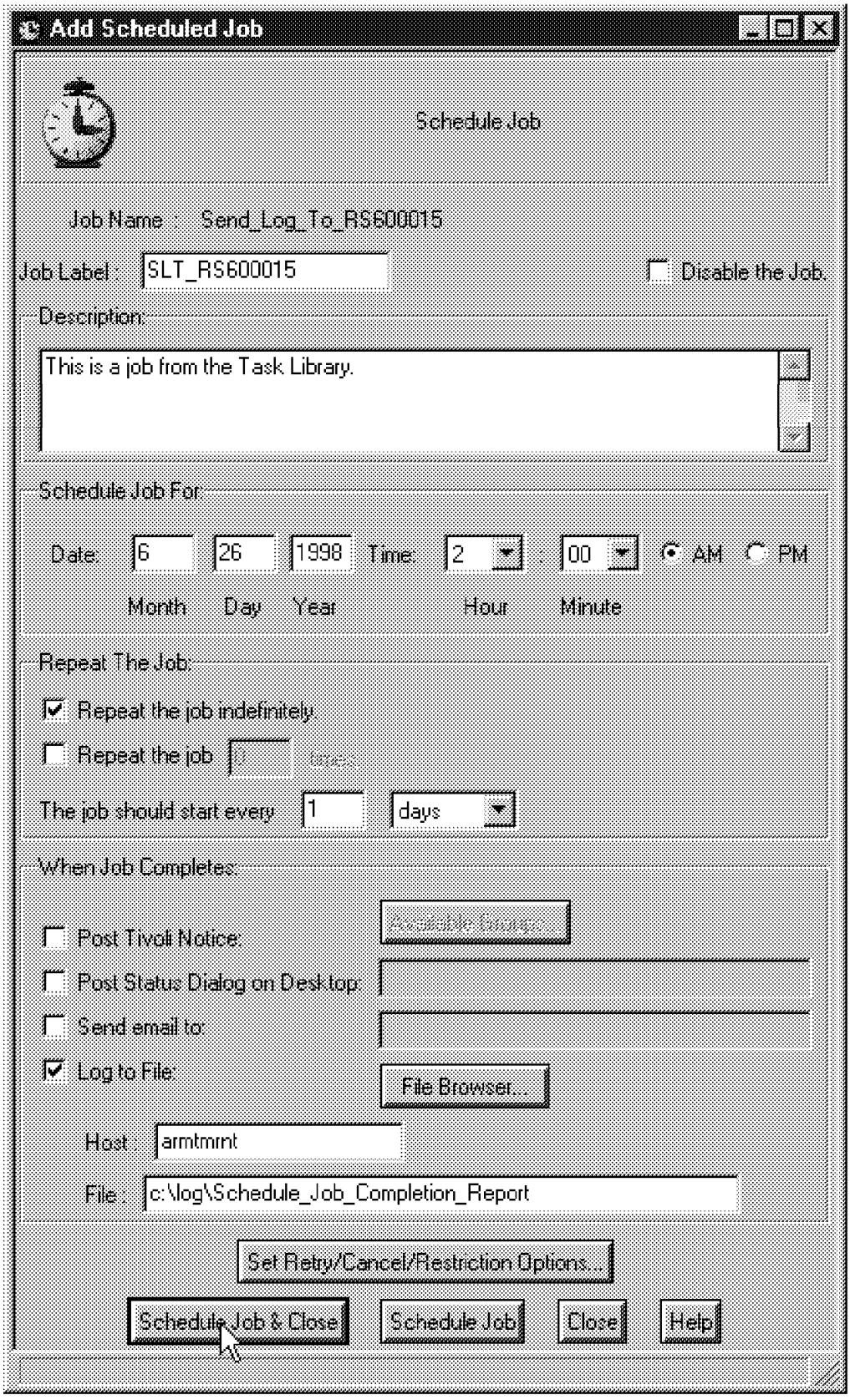


Figure 73. Filling in the Schedule Job Dialog Box

In the Repeat The Job section we checked the **Repeat the job indefinitely** box and set it to repeat every day.

Now move to the When Job Completes section. We decided to have a file created with output messages.

Finally, click on **Schedule Job & Close**.

4. To verify that your job has been scheduled, double click on the **Scheduler** icon which is located on the Tivoli Desktop.

   If a job is currently scheduled you will see a screen similar to Figure 74.

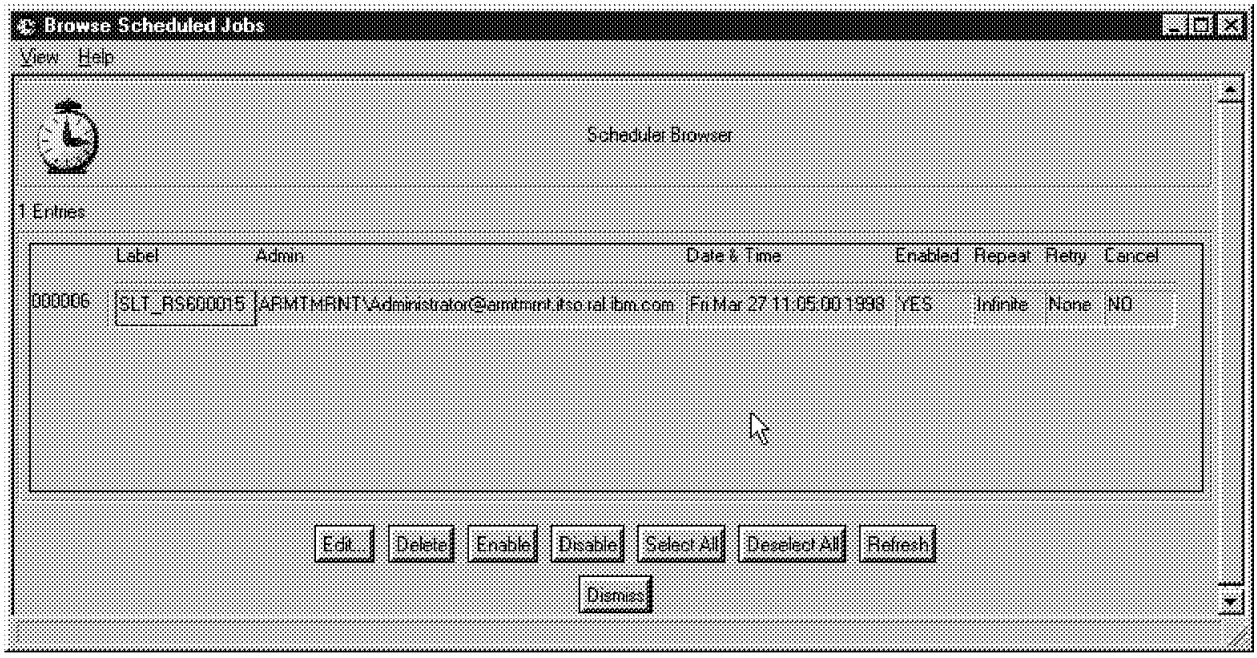


Figure 74. The Scheduled Job

When the scheduled job ran, we found two new files in the Log directory on our machine. If you use Wordpad to open these files you will find information about the completion status of the job, as you can see in Figure 75 on page 98.

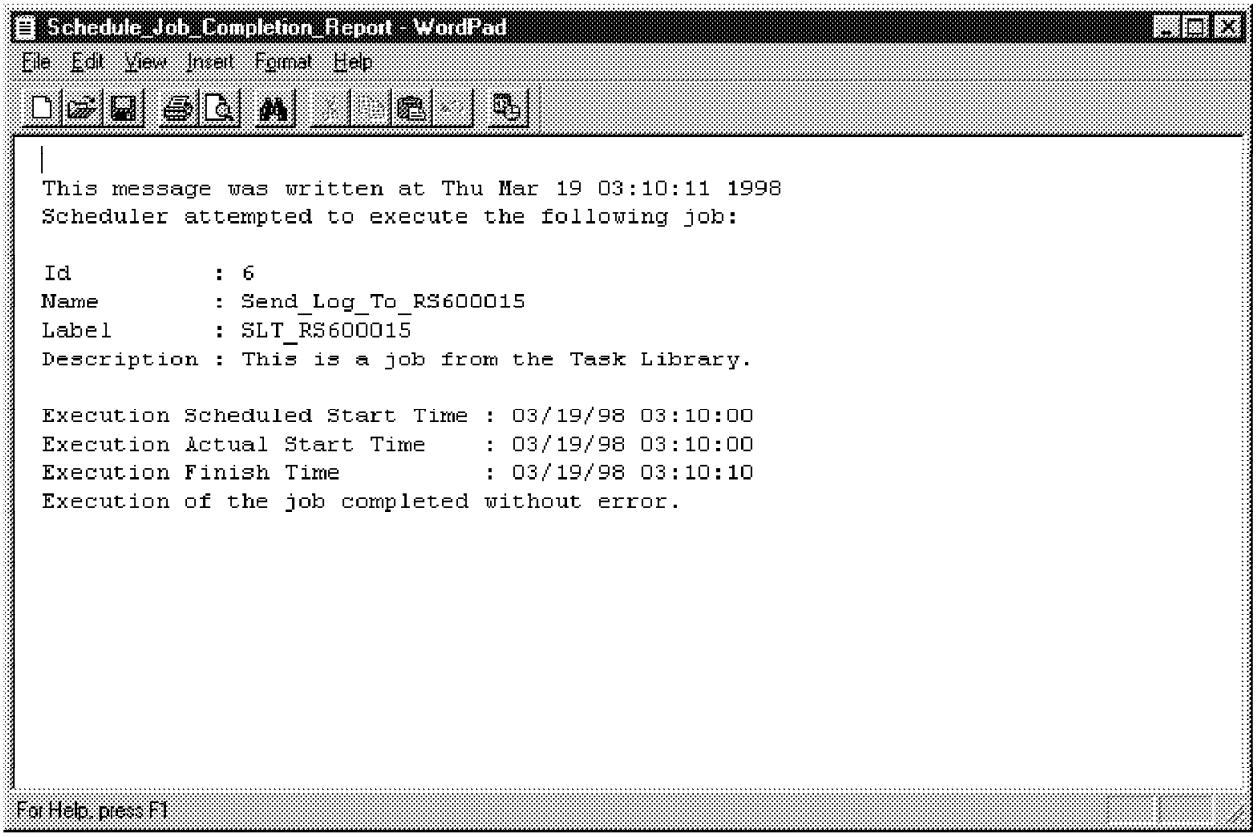


Figure 75. The Completion Status of the Scheduled Job

7.6 Automating the Startup of Servers, Clients and Collections

We wanted to automatically start all the ARM clients that connect to a server, immediately after the server starts. In order to do this, we defined an automated task to be run each time the server starts.

We completed the following steps:

1. Modify the StartARMServer task to cause an event each time the ARM Server Agent is started.
2. Create an automated task to be run when this event occurs.

7.6.1 Modifying the StartARMServer Task

To get the StartARMServer task to send an event after the server is started, we edited the file that is executed when the task is run.

On our Windows NT TMR server, this file was located in:

`\Tivoli\bin\platform\Tas\TASK_LIBRARY\bin\some_number`

where

- `platform` is the target operating system
- `some_number` is a combination of numbers and letters

Depending on the platform, this file may be a script or a batch file. The name of this file on our machine was:

```
ARM_Library__xgxffufa
```

Here is our file, with the changes highlighted:

```
@echo OFF
setlocal
set PARAMS=
if %1 == "" goto SENTRY
set PARAMS= -f %1
:SENTRY
if %2 == " " goto CMND
set PARAMS=%PARAMS% -s
:CMND
CALL %WINDIR%\system32\drivers\etc\tivoli\setup_env.cmd
@echo OFF
set CHILD_OF_OSERV=
warmcmd srvstart %PARAMS%
if not errorlevel 1 goto SNDMSG
goto END
:SNDMSG
wpostemsg -r WARNING -m "server started" EVENT NT
:END
EXIT
```

The errorlevel variable contains the return code of the previous command run by cmd.exe (in this case, warmcmd).

The line

```
if not errorlevel 1 goto SNDMSG
```

specifies that if the value of *errorlevel* is not greater than or equal to one (in other words the return code is zero), control should pass to SNDMSG.

The wpostemsg command sends an event to the event server in the Tivoli Enterprise Console (TEC). The arguments on this command are as follows:

- **-m** specifies that the message should be sent to the event server.
- **-r** specifies the severity of the event; we chose *WARNING*.
- **EVENT** is the class of the event - it must match an event class defined in the event server.

  To see the event classes defined in the event server:

  1. Click with the right mouse button on the **Event Server** icon, and then select **EventGroups**.
  2. In the Event Groups window, select the group you are interested in or **All**, and then click on **Edit** on the toolbar, and select **Edit Filters**.
  3. In the Edit Event Group Filters window, click on the **Event Class** button, and you can see all the event classes defined for this event group.
  4. Now click on **Close** to return.
- **NT** is the source of the event. It must match a source configured at the server.

To see which sources are defined in the server, open the TEC Display and two windows will appear. In the Source Group window, you can see all the sources defined in the event server.

7.6.2 Creating an Automated Task to be Run When an Event Occurs

Now we know that an event will occur when the ARM Server Agent is started. So we need to set up a task to run automatically when the event occurs.

To trigger an automated task on a event, open the TEC console display. Two windows will appear: the Source Group window and the Event Group window. Click on the **NT** icon in the Source Group window, so that the Event window for NT opens.

To define an automated task, follow these steps:

1. Click on **Automated Task** on the toolbar, followed by **New**.
2. In the Select Event Class window, select the **EVENT** event class.
3. Click on **Select & Close**. The Summary of the Automated Tasks window will appear.
4. You can edit the criteria when to execute the task. Click on **Edit Criteria** to define the conditions that the event must match, to run the task.

   Or you can choose not to set any criteria, so the task will execute anytime the event arrives.
5. Now click on **Add Task**.
6. In the window that appears, double-click on **ARM Library** and in the pane on the right, you will see the list of tasks defined in the ARM Library.
7. Double click on the **StartARMClient** task.
8. Fill in the hostname of the ARM Server Agent machine that this client will connect to.
9. At the bottom of this window, select **Execute on Select Managed Nodes** and then click on **Add & Close**.
10. Now select the managed node where the task should execute and click on **Set**.
11. Now the startARMClient task has been added to run on the first client. Repeat the steps to add the startARMClient task for all the clients.
12. Click on **Save As** and give a name to the automated task you′ve created, and then click on **Close**.

Now, you have an automated task defined. When the EVENT occurs and the criteria you′ve defined matches the event, the startARMClient task is executed on all your clients.

To get all of this to work properly, TEC must be running, so you should see the Event Group and Source group windows on the Desktop.

Chapter 8. Programming with the ARM API

If you want the ARM Agents to measure response times for the applications that you are using, those applications must be instrumented to the ARM API.

In this chapter we discuss what this means, and show how to instrument some sample applications written in C, PowerBuilder and Visual Basic.

8.1 Using the ARM API Calls in an Application

As we said in 1.1, "Instrumenting Applications for Response Time Measurement" on page 1, to instrument an application for ARM, you must place ARM API calls at appropriate points in the application code.

You can see in Figure 76 an overview of how these calls should be placed within the application code.

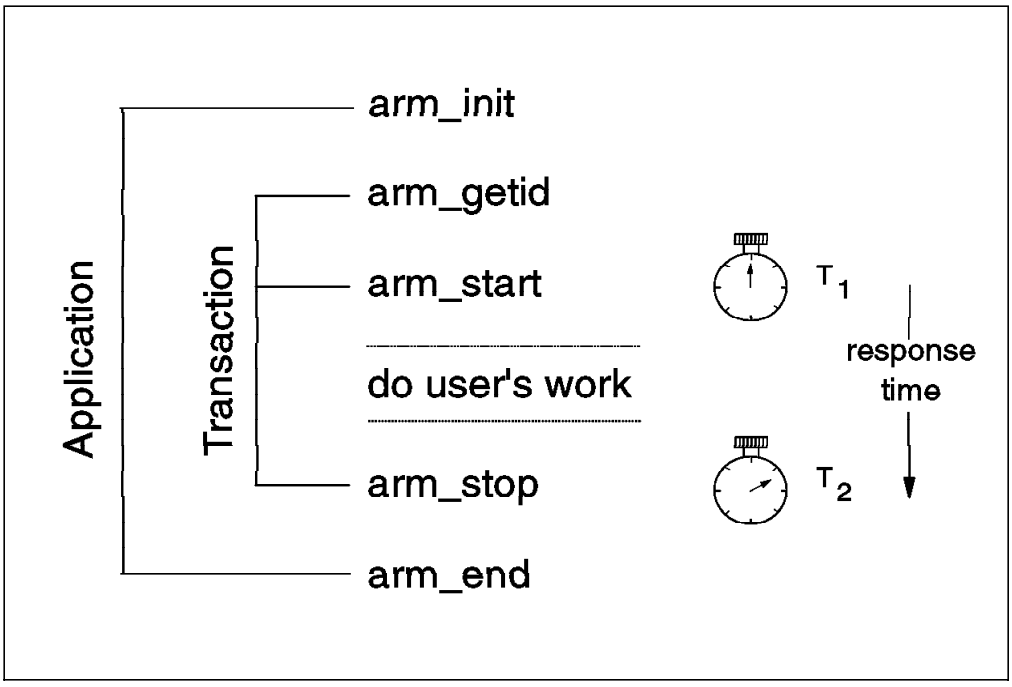


Figure 76. The Relationship Between ARM API Calls

8.1.1 The ARM API Call Syntax

Here we briefly discuss each of the ARM API calls. We use example calls from the ARMTEST.C application listed in Appendix A, "Source File Listings" on page 115.

8.1.1.1 arm_init

This call is used to define an application. It must be made before any of the other ARM API calls related to that application.

```
appl_handle = arm_init(application,"*",0,0,0);
```

We defined our application with a name of *application*, and set the appl_user_id field to *, so that when Distributed Monitoring reports on this application, it includes the login user ID of the person running it; this is looked up from the operating system.

The handle returned by this call will be placed in the variable called *appl_handle*, and this will be used on the arm_getid call.

8.1.1.2 arm_getid

This call is used to define a transaction, which must be a child of an application.

```
getid_handle = arm_getid(appl_handle,transaction,"W95/NTwks",0,0,0);
```

We defined our transaction with a name of *transaction*, picked up the variable *appl_handle* that was returned by the arm_init call, and set the tran_detail field to W95/NTwks, so that when Distributed Monitoring reports on this transaction, it can include this information indicating the type of machine where it is running.

The handle returned by this call will be placed in the variable called *getid_handle*, and this will be used on the arm_start call.

8.1.1.3 arm_start

You make this call when the transaction starts running. It starts the response time clock.

```
start_handle = arm_start(getid_handle,0,0,0);
```

The call uses the variable *getid_handle* that was returned by the arm_getid call.

The handle returned by this call will be placed in the variable called *start_handle*, and this will be used on the arm_update call, if it is made, and on the arm_stop call.

8.1.1.4 arm_update

This call is optional. It can be used as a heartbeat, to check the progress of a long-running transaction. It can be used as many times as you want, between the arm_start and arm_stop calls. We did not use the arm_update call in our ARMTEST.C program, but if we had, it would have looked like this:

```
updaterc = arm_update(start_handle,0,0,0);
```

The call uses the variable *start_handle* that was returned by the arm_start call.

This call returns an error code, rather than a handle, which will be placed in the variable called *updaterc*.

8.1.1.5 arm_stop

You make this call when the transaction stops running. It stops the response time clock.

```
stoprc = arm_stop(start_handle,ARM_GOOD,0,0,0);
```

The call uses the variable *start_handle* that was returned by the arm_start call.

We set the transaction status field to *ARM_GOOD*, for simplicity. The other options are *ARM_ABORT* and *ARM_FAILED* and these can be used by Distributed Monitoring to create alarms if the transaction does not complete the way you would like it to.

This call returns an error code, rather than a handle, which will be placed in the variable called *stoprc*.

8.1.1.6 arm_end

This call should be made when the application terminates. It causes a cleanup of the memory that the ARM Agent has allocated for this application.

```
endrc = arm_end(appl_handle,0,0,0);
```

The call uses the variable *appl_handle* that was returned by the arm_init call.

This call returns an error code, rather than a handle, which will be placed in the variable called *endrc*.

8.1.2 Case Sensitivity

The ARM API calls are C functions, and are therefore case sensitive.

All the ARM API functions use lowercase throughout. If you code any part of the API calls in uppercase,

- If you are programming in C or C++, then the linker will fail to resolve the external functions.
- If you are using a language other than C, the call will fail with a return of -1.

8.1.3 The Returns from ARM API Calls

Each of the ARM API calls produces a return. In every case, this is 32-bit in size, and is signed.

The nature of the return differs according to the call.

The following calls return handles, which are used on subsequent calls:

- arm_init
- arm_getid
- arm_start

The following calls return integer return codes:

- arm_update
- arm_stop
- arm_end

8.1.3.1 Handles and Integers

You must declare integer variables in your program, to hold the handles and return codes from the ARM API calls, but the handles *cannot* be treated as integer variables.

An integer is a numeric value on which numeric operations may be performed, whereas a handle is a system-assigned ″shortcut″ to reference something which the system has created.

The handle has meaning only to the ARM Agent, and it cannot be modified in any way.

8.1.3.2 Good handles/return codes

When your application makes successful ARM API calls, you can expect to receive the following returns. In the case of the *arm_init*, *arm_getid* and *arm_start* calls, the return will be used as the handle.

arm_init	positive integer
arm_getid	positive integer
arm_start	positive integer
arm_update	zero
arm_stop	zero
arm_end	zero

8.1.3.3 Bad Handles and Return Codes

Any of the API calls can produce the following returns:

-1 An invalid argument was specified.

Usually this means that you made a mistake in the syntax of the call. For example, you may have coded too many, or too few arguments on the call.

If you are programming in C or C++, such errors would be caught by the linker, when it attempts to resolve the external references in the shared library that is shipped with the agent.

If you are calling the ARM API as external functions from another language, such as PowerBuilder, there is no link step, and so it's possible for such a mistake to slip by, and you will only be aware of it when the ARM API call is actually made.

-2 The Client Agent is not running.

This error usually occurs because the Client Agent has not been started on the machine where you are making ARM API calls.

It's not enough just to have the agent installed; you must actually start it running too.

-3 Communication with the Client Agent cannot be established.

The arm_init, arm_getid and arm_start calls can additionally produce the following return.

-4 An error was found in the Client Agent. An error message is stored in the log file.

8.1.4 Using ARM API Calls Efficiently

The best way to place ARM calls in your program is to put all the arm_init and arm_getid calls at the top of the source code, just as you would declare all variables at the start of a program. This means that the definition of all the applications and transactions will be performed at program startup.

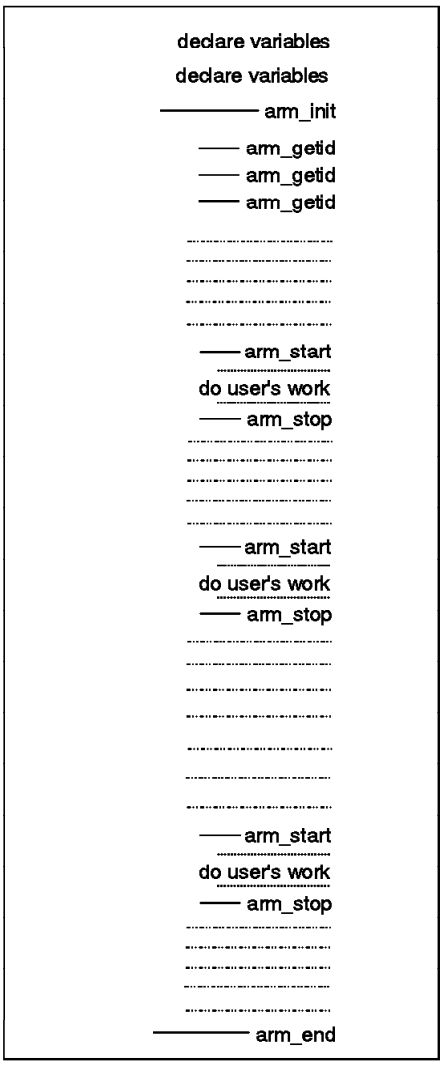


Figure 77. Making the arm_init and arm_getid Calls at Program Initialization

8.1.5 Using the arm_update API Call

The arm_update call is optional. It is most useful in a transaction that is likely to have a long running time.

You may want to use it for the following purposes:

- It can provide a heartbeat, to indicate that a transaction is still running.
- It can inform you of the progress of a transaction, so that you can see how much of the transaction′s work has been completed.
- If a transaction hangs, it can be used to indicate the point at which the hang occurred.

You can see how the arm_update call can be used multiple times in the application shown in Figure 78 on page 106.

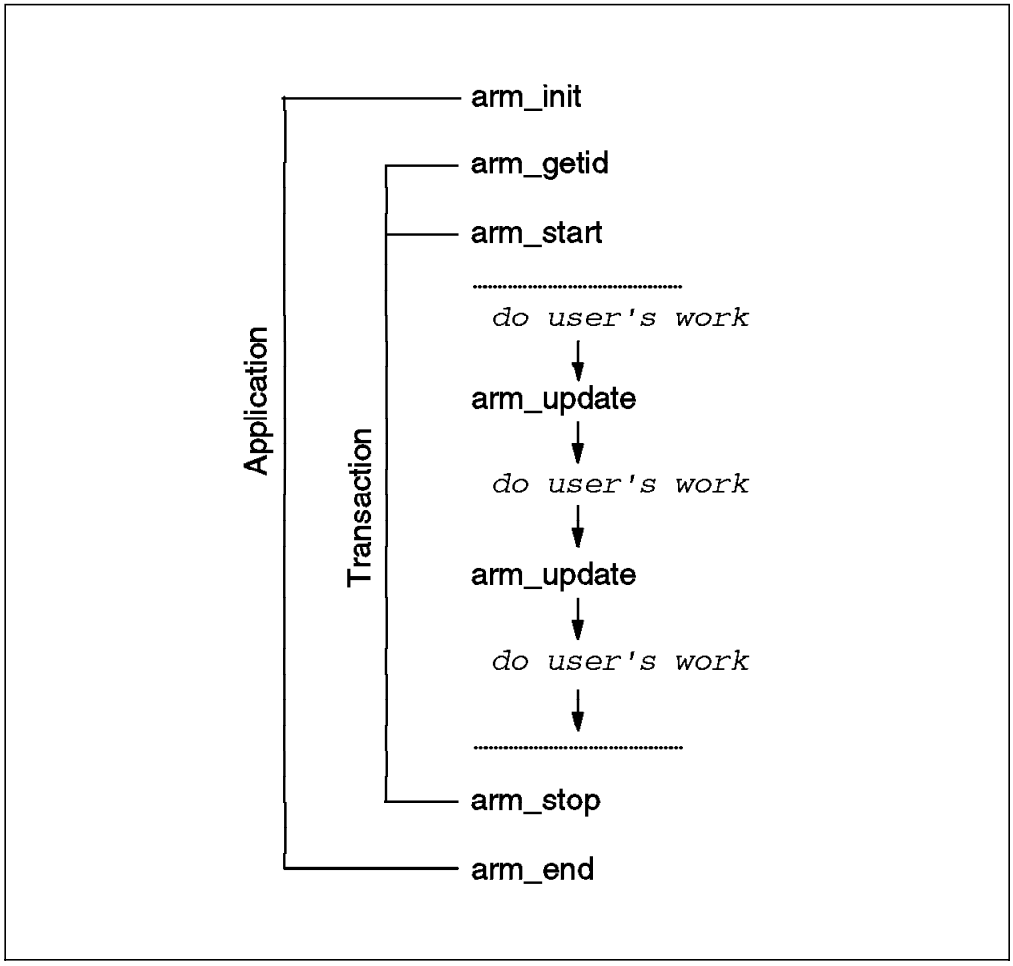


Figure 78. arm_update Calls in an Application

8.2 Compiling with ARM

Once you have placed the ARM API calls in your C source file, you are ready to compile.

The compile process for an ARM-instrumented application is shown in Figure 79 on page 107. Notice that you will need the header and library files for ARM, to provide the prototypes for the ARM functions, and then resolve those functions during the link step.

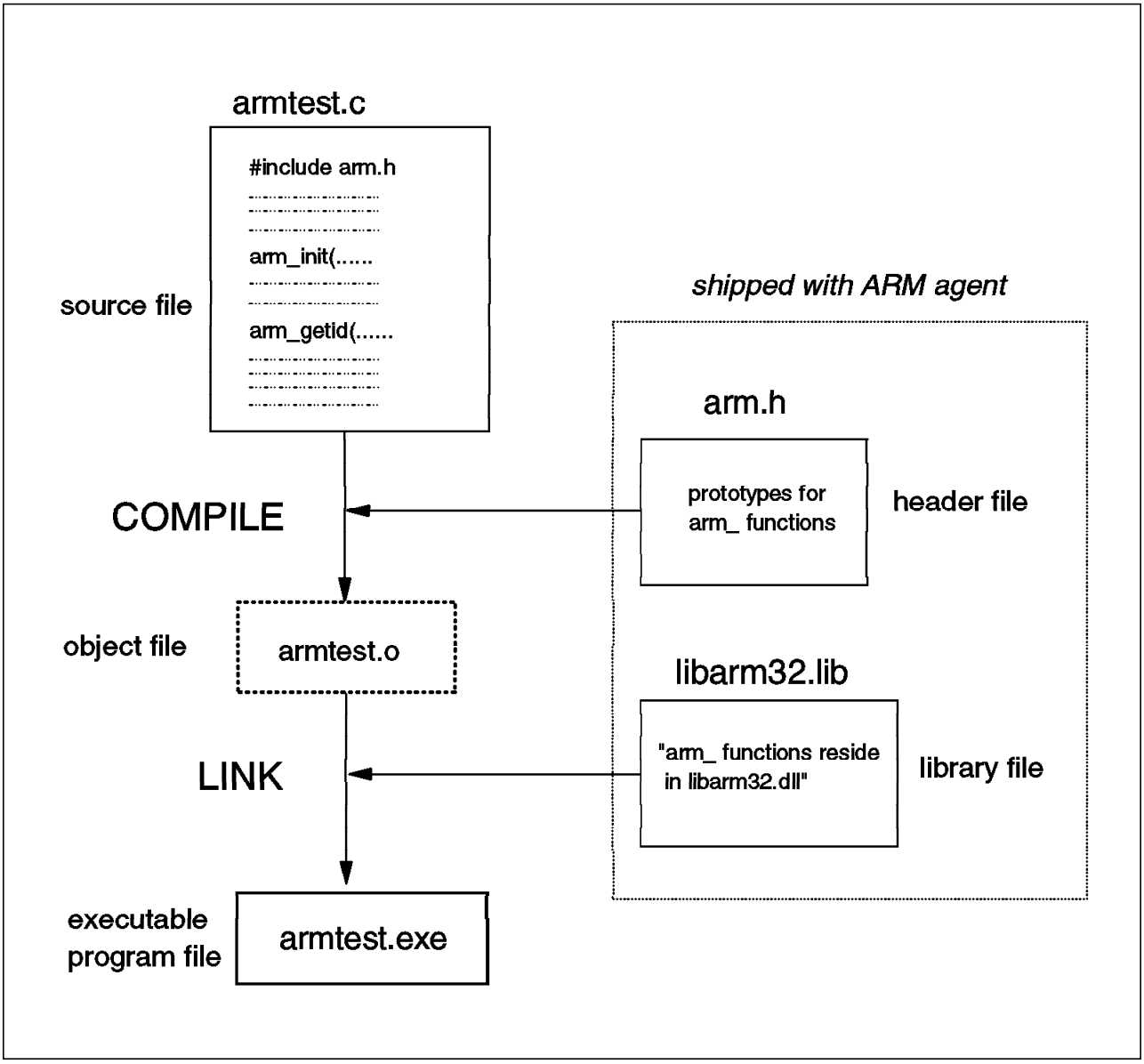


Figure 79. The Compile and Link Process for an ARM-Instrumented Application

Before compiling, you may want to check the following:

1. The required header and library files are present on the machine. You will need the header and library files appropriate for the operating system platform on which the application will run.

   You do not need to have an ARM Agent installed or active on the machine where you are compiling.

2. The directory that contains the header and library files has been specified in the compiler's *include* and *link* directory paths.

3. You have specified the correct library file to the linker.

8.3 Compiling 16-bit Applications for Windows 95 and NT

Windows 95 and Windows NT can support both 16-bit and 32-bit applications natively.

If you instrument a 32-bit application to make ARM API calls, these will be happily received by the 32-bit ARM Agent.

If you instrument a 16-bit application for ARM, and want to run it on Windows 95 or NT, you will need to use the following files when you compile it:

- arm16.h
- libarm16.lib

When you run your application and it makes ARM API calls, it will load libarm16.dll, which will in turn load libarm32.dll, so make sure that you have both these DLLs available at runtime, either in the same directory as your application, or in the Windows directory.

This situation is summarized in Figure 80.

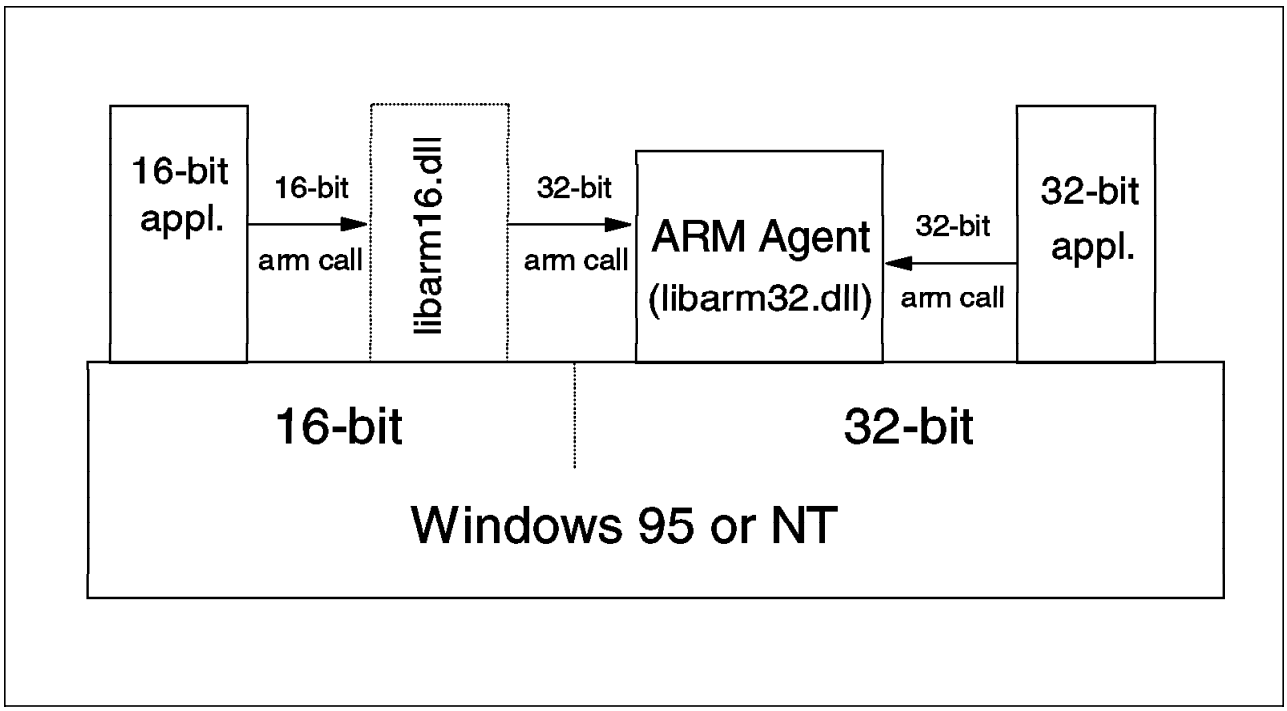


Figure 80. Running ARM-Instrumented 16-bit and 32-bit Applications on Windows 95 or NT

8.4 Using Languages Other Than C

The ARM API can be called from any language, but the agents ship today with header and library files for the C and C++ languages.

If you are using a language other than C or C++, we can think of two ways of making calls to the ARM API:

1. Call the ARM API as an external C function from your programming language

2. Convert the header and library files into a format suitable for your language.

8.4.1 Making ARM API Calls From PowerBuilder Applications

We used a PowerBuilder application as an example of how to call the ARM API using a language other than C or C++.

PowerBuilder provides its own development language, called PowerScript. You can make all the ARM API in PowerScript, following its syntax rules.

Here is an example, from a PowerBuilder script, of how the ARM API calls can be made. After each call, this script pops up a window to display the handle, which can be very useful during initial development and testing.

```
arm_application = arm_init("new_account", "*",0,0,0)
messagebox("arm_init", string(arm_application))

arm_transaction = arm_getid(arm_application, "open", "  ", 0,0,0)
messagebox("arm_getid", string(arm_transaction))

arm_start_val = arm_start(arm_transaction, 0,0,0)
t = getapplication()
messagebox("arm_start", string(arm_start_val))


t.af_opencustomer(s)

arm_stop = arm_stop(arm_start_val, 0,0,0,0)
messagebox("arm_stop", string(arm_stop))

arm_end_val = arm_end(arm_transaction,0,0,0)
messagebox("arm_transaction", string(arm_end_val))
```

Figure 81. ARM API Calls in a PowerBuilder Script

The script calls the ARM API functions. In order for these function calls to be resolved, you will need to declare the ARM API functions to PowerBuilder, as Global External Functions.

To do this, click **Declare** on the toolbar, in a PowerBuilder window, and open the Global External Function editor. Now you can fill in the ARM function prototypes, exactly as in Figure 82. Note that these declarations are case sensitive; everything must be in lowercase.

If you are accessing this book in softcopy, for example on the Internet, or from a CD, we suggest that you copy and paste the definitions, to avoid the possibility of typing errors.

```
function long arm_init(string name,string user_id,long flag,long data,long size) library "libarm32.dll"
function long arm_getid(long id,string name,string detail,long flag,long data, long size) library "libarm32.dll"
function long arm_start(long id,long flag,long data,long size) library "libarm32.dll"
function long arm_update(long handle,long flag,long data,long size) library "libarm32.dll"
function long arm_stop(long handle,long status,long flag,long data,long size) library "libarm32.dll"
function long arm_end(long id,long flag,long data,long size) library "libarm32.dll"
```

Figure 82. Declaring the External Functions to PowerBuilder

Now, when the Powerbuilder application runs, the ARM calls that it makes will be resolved in the external function definitions, and will result in ARM API calls being made to the ARM Client Agent, as you can see in Figure 83 on page 110.

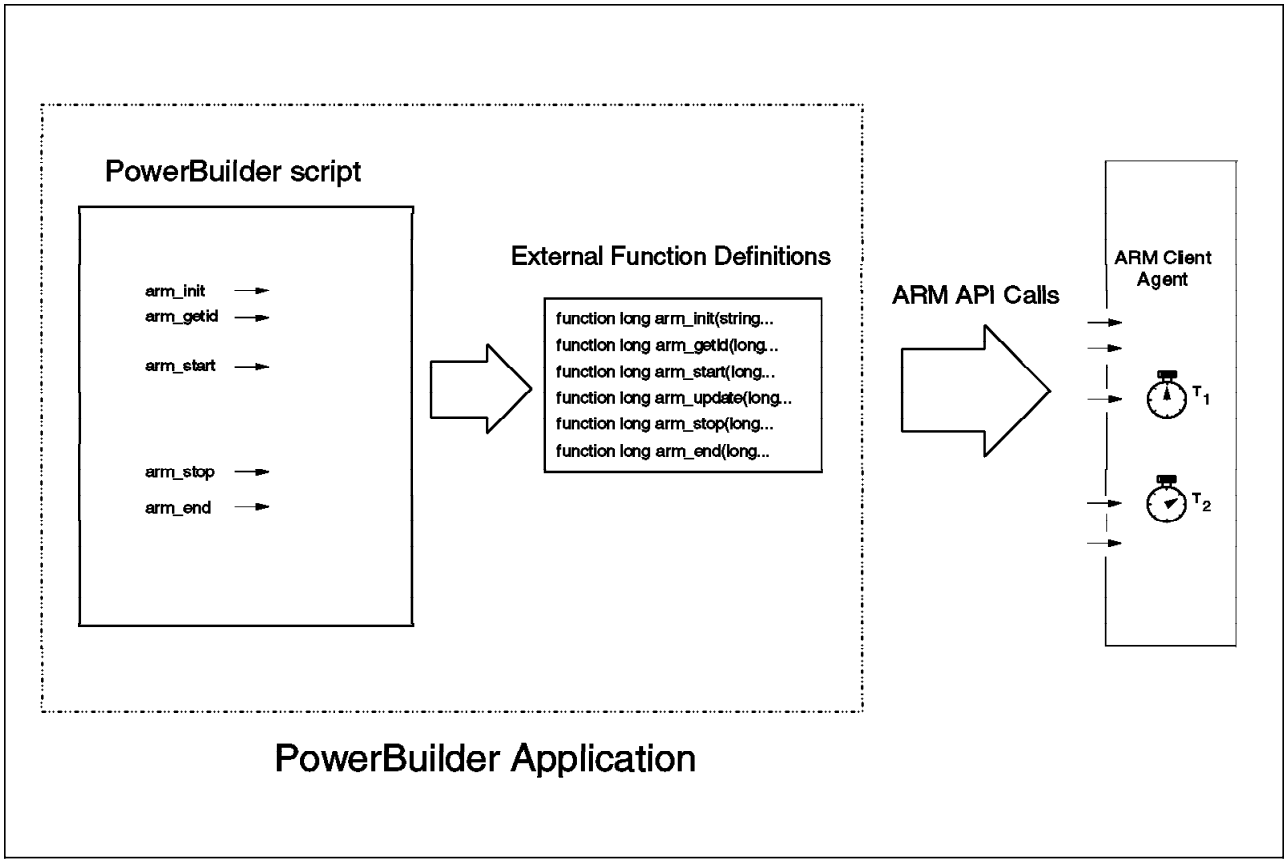


Figure 83. Resolving the ARM Calls as PowerBuilder External Functions

8.4.2 Making ARM API Calls From Visual Basic Applications

We wanted to make ARM API calls from a Visual Basic application. There are two steps involved in doing this:

1. Declare the ARM API functions to Visual Basic as DLL procedures.
2. Call the DLL procedures in your Visual Basic program.

8.4.2.1 Declaring the ARM API Functions as DLL Procedures

You only need to declare the ARM API functions once; then you can call them as many times as you like in your Visual Basic program.

We declared the DLL procedures in a standard module, so that they were public, and could be called from any part of our application.

The required declarations are shown in Figure 82 on page 109. Note that they are case sensitive; everything must be in lowercase.

If you are accessing this book in softcopy, for example on the Internet, or from a CD, we suggest that you copy and paste the declarations, to avoid the possibility of typing errors.

```
Declare Function arm_init Lib "libarm32.dll" (appl_name As Variant,appl_user As Variant,
ByVal flags As Long,data As Variant,ByVal data_size As Long) As Long

Declare Function arm_getid Lib "libarm32.dll" (ByVal handle As Integer,tran_name As Variant,
tran_detail As Variant,ByVal flags As Long,data As Variant,ByVal data_size As Long) As Long

Declare Function arm_start Lib "libarm32.dll" (ByVal handle As Integer,ByVal flags As Long,
data As Variant,ByVal data_size As Long) As Long

Declare Function arm_update Lib "libarm32.dll" (ByVal handle As Integer,ByVal flags As Long,
data As Variant,ByVal data_size As Long) As Integer

Declare Function arm_stop Lib "libarm32.dll" (ByVal handle As Integer,ByVal tran_status As Long,
ByVal flags As Long,data As Variant,ByVal data_size As Long)As Integer

Declare Function arm_end Lib "libarm32.dll" (ByVal handle As Integer,ByVal flags As Long,
data As Variant,ByVal data_size As Long) As Integer
```

Figure 84. Declaring the ARM DLL Procedures to Visual Basic

8.5 Using the ARM Software Developer's Kit

The ARM Software Developer's Kit (SDK) allows you to build and run application programs that are instrumented for ARM, even if you do not yet have a real ARM Agent.

You can download a copy of the SDK for free from the Tivoli downloads area at http://www.tivoli.com/.

The SDK includes the header and library files necessary to compile an ARM-instrumented application that is written in C or C++, and a no-operation (NOP) runtime library.

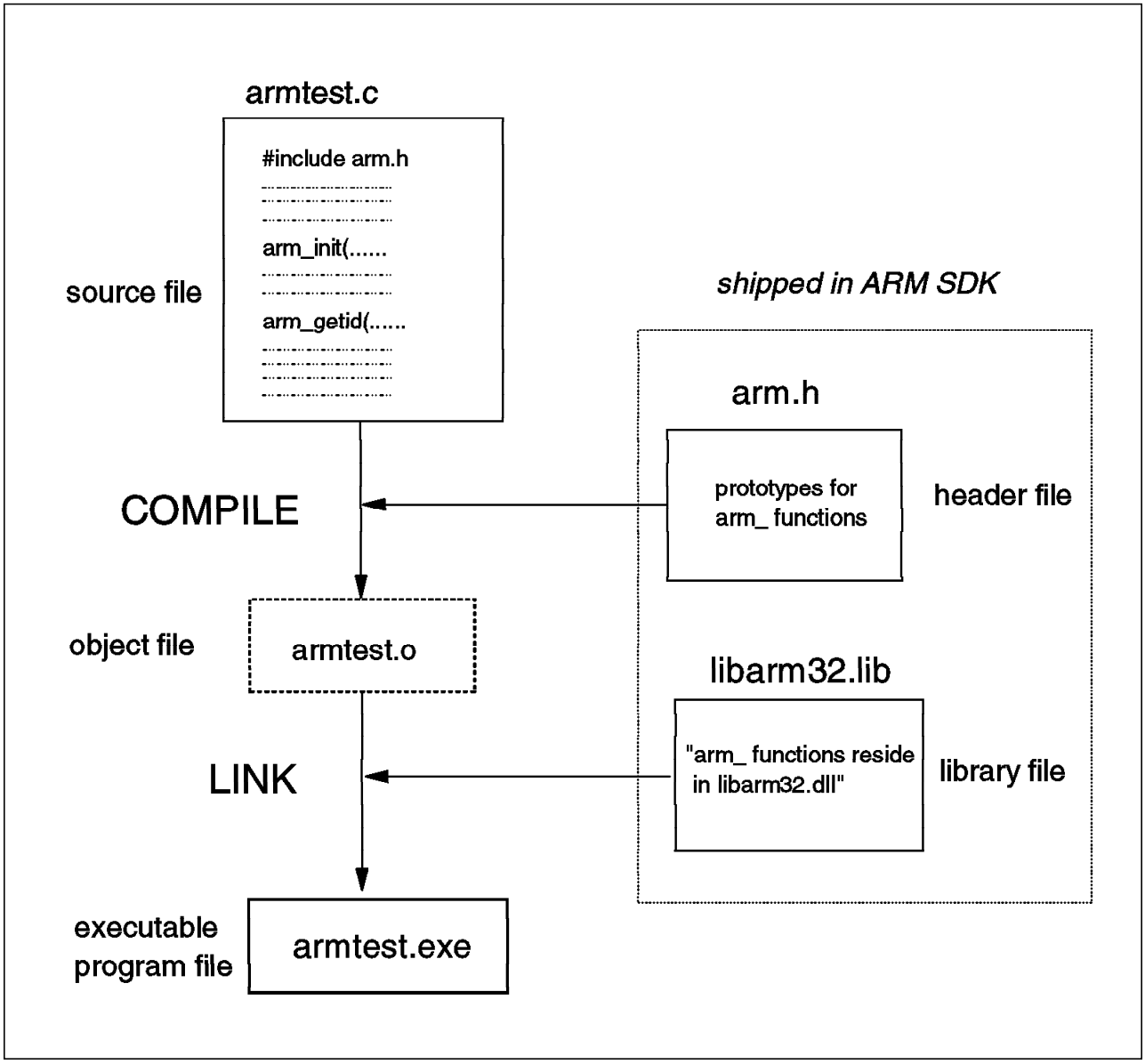


Figure 85. Using the ARM SDK to Compile and Link a Windows 95/NT Program

The NOP library will receive the ARM API calls that your application makes when it runs, and return a handle of zero.

When you do install a real ARM Agent, you should just need to replace the NOP with the real run-time library that the agent ships.

Bug in the Windows Agent

We found, in the early Limited Availability code, that a few of the file names differ between the SDK and the Tivoli ARM Agent. We have *highlighted* those that differ in Table 2 on page 113.

This problem is currently being addressed by development, and should be fixed in the GA code.

Table 2. File Names in the ARM SDK and the Tivoli ARM Agents

Operating System	SDK			Tivoli ARM Agent		
	Header	**Library**	**Runtime**	**Header**	**Library**	**Runtime**
Windows 95/NT (32-bit apps.)	arm.h	libarm32.lib	libarm32.dll	arm.h	*libarm.a*	*libarm.dll*
Windows 95/NT (16-bit apps.)	N/A	N/A	N/A	arm16.h	libarm16.lib	libarm16.dll
Windows 3.1	arm.h	libarm.lib	libarm.dll	N/A	N/A	N/A
AIX	arm.h	libarm.a	libarm.a	arm.h	libarm.a	libarm.a
HP-UX	arm.h	libarm.sl	libarm.sl	arm.h	libarm.sl	libarm.sl
Solaris	arm.h	libarm.so	libarm.so	arm.h	libarm.so	libarm.so

One thing to think about, if you are considering developing an ARM-instrumented application with the SDK, is that the SDK′s runtime library will always return a zero when your application makes an ARM API call, even if the call is invalid.

In contrast, the Tivoli ARM Agent will return error codes in the handle.

For example, you could make an arm_stop call in your program, for a transaction that was never started. The SDK agent will return a zero, as if everything is OK, whereas the Tivoli agent will recognize that you made an arm_stop call for which there was no corresponding arm_start call.

Therefore, we strongly recommend that you test any ARM-instrumented application with a real ARM Agent, before putting the application into production.

Appendix A. Source File Listings

Here we have included the source code for two simple interactive line mode programs that are instrumented for the ARM API. This should help you see how the API calls are used. The source code is available from our FTP server; see Appendix B, “How to Get the Samples Used in this Book” on page 121 for details.

These programs are very useful for testing the ARM agents, once you have them installed.

Both programs are written for a Windows 32-bit environment, but are easily modified for other platforms by commenting the following lines:

```
#define  W32

#include <windows.h>
```

A.1 SIMPARM.C

This is a very basic program, to show the use of the ARM API calls. It is used as the basis for the more sophisticated program ARMTEST.C. We have highlighted the ARM API calls in bold type, for illustrative purposes.

```
/*****************************************************************************/
/* This is a simple interactive demonstration program.                       */
/* It tests the ARM Agents for Windows 95 and NT Workstation.                */
/* The agents are part of Distributed Monitoring.                            */
/*****************************************************************************/

#define  W32

#include <windows.h>
#include "arm.h"
#include <stdio.h>
#include <string.h>


void main()
/*****************************************************************************/
/* Define the variables                                                      */
/*****************************************************************************/
{

  char application[20];
  char transaction[20];
  long appl_handle;
  long getid_handle;
  long start_handle;
  char string[20];
  long stoprc;
  long endrc;


  /*****************************************************************************/
```

```
  /* Ask for application name and make arm_init call                          */
  /***************************************************************************/

  printf("\nThis program generates transactions for the ARM Agents of\n");
  printf("Distributed Monitoring, on Win 95 and NT workstation.\n\n");

  printf("Please enter the name of your application:\n");
  scanf("%s",&application);

  appl_handle = arm_init(application,"*",0,0,0);

  /***************************************************************************/
  /* Ask for transaction name and make arm_getid call                        */
  /***************************************************************************/

  printf("\nPlease enter the name of your transaction:\n");
  scanf("%s",&transaction);

  getid_handle = arm_getid(appl_handle,transaction,"W95/NTwks",0,0,0);

     /************************************************************************/
     /* Start transaction and make arm_start call                            */
     /************************************************************************/

     start_handle = arm_start(getid_handle,0,0,0);

     printf("\nTransaction started...\n");
     printf("Type some characters and press ENTER when\n");
     printf("you want to stop the transaction.\n");
     scanf("%s",&string);

     /************************************************************************/
     /* Stop transaction and make arm_stop call                              */
     /************************************************************************/

     printf("\n\nTransaction stopped.\n\n");
     stoprc = arm_stop(start_handle,ARM_GOOD,0,0,0);

  /***************************************************************************/
  /* End application by making arm_end call                                  */
  /***************************************************************************/

  endrc = arm_end(appl_handle,0,0,0);
}

/*****************************************************************************/
/* End of program                                                            */
/*****************************************************************************/
```

A.2 ARMTEST.C

This program was developed from SIMPARM.C.

It includes a loop, so that the transaction can easily be repeated many times, and it presents the returns from the ARM API calls to the user, with explanations of their meaning. We have highlighted the ARM API calls in bold type, for illustrative purposes.

```
/*************************************************************/
/* This is a simple interactive demonstration program.       */
/* It tests the ARM Agents for Windows 95 and NT Workstation.  */
/* The agents are part of Distributed Monitoring.            */
/*************************************************************/

void showError(char* func);     /* Prototype for showError function */

#define  W32

#include "arm.h"
#include <windows.h>
#include <stdio.h>
#include <string.h>

int arm_rc;     /* Global variable */


void main()

/*********************************************************/
/* Define the local variables                            */
/*********************************************************/
{

  char application[20];
  char transaction[20];
  int  appl_handle;
  int  getid_handle;
  char startcmd[20];
  int  start_handle;
  char endcmd[20];
  int  stoprc;
  int  endrc;


  printf("\nThis program generates transactions for the ARM Agents of\n");
  printf("Distributed Monitoring, on Win 95 and NT workstation.\n\n");


  /*********************************************************/
  /* Ask for application name and make arm_init call       */
  /*********************************************************/

  printf("Please enter the name of your application:\n");
  scanf("%s",&application);

  appl_handle = arm_init(application,"*",0,0,0);

  arm_rc=appl_handle;
  showError("arm_init");    /* Call the showError function */


  /*********************************************************/
  /* Ask for transaction name and make arm_getid call      */
  /*********************************************************/

  printf("Please enter the name of your transaction:\n");
  scanf("%s",&transaction);
```

```
getid_handle = arm_getid(appl_handle,transaction,"W95/NTwks",0,0,0);

arm_rc=getid_handle;
showError("arm_getid");    /* Call the showError function */


      /*********************************************************/
      /*  Prompt user to start the transaction                 */
      /*********************************************************/

      do {
          printf("Type \"start\" to start the transaction\n");
          scanf("%s",&startcmd);

      /*********************************************************/
      /*  Test whether user typed start                        */
      /*********************************************************/

      } while (strcmp(startcmd,"start"));



/*********************************************************/
/* Loop to repeat transactions                           */
/*********************************************************/

do {

   /*********************************************************/
   /* Make arm_start call at transaction start              */
   /*********************************************************/

   start_handle = arm_start(getid_handle,0,0,0);

   printf("\nTransaction started...\n");

   arm_rc=start_handle;
   showError("arm_start");    /* Call the showError function */

   printf("Type a few characters and press ENTER.\n");
   printf("Or type \"end\" and press ENTER, to end the program.\n\n");
   scanf("%s",&endcmd);


   /*********************************************************/
   /* Make arm_stop call at transaction end                 */
   /*********************************************************/

   stoprc = arm_stop(start_handle,ARM_GOOD,0,0,0);

   printf("\nTransaction stopped.\n");

   arm_rc=stoprc;
   showError("arm_stop");    /* Call the showError function */

} while (strcmp(endcmd,"end"));
```

```
  /*********************************************************/
  /* Make arm_end call at application end                  */
  /*********************************************************/

  endrc = arm_end(appl_handle,0,0,0);

  arm_rc=endrc;
  showError("arm_end");    /* Call the showError function */

}


/*****************************************/
/* Function for reporting return codes       */
/*****************************************/

void showError(char* func)
{
    if (arm_rc > -1)
    {
        printf("The return from your %s call is: %d \t[Good!]\n\n",func,arm_rc);
    }
    else
    {
        printf("The return from your %s call is: %d",func,arm_rc);

        switch (arm_rc)
        {
            case -1  :
              printf("\n[ERROR: An invalid argument was used in the call]\n\n");
              break;

            case -2  :
              printf("\n[ERROR: The ARM agent is not running]\n\n");
              break;

            case -3  :
              printf("\n[ERROR: Communication with the agent cannot \n");
              printf("be established]\n\n");
              break;

            case -4  :
              printf("\n[ERROR: An error was found in the agent; \n");
              printf("look in the log file]\n\n");
              break;

            default  : printf("\n[ERROR: Unknown error code]\n\n");
              break;
        }
    }
}


/*********************************************************/
/* End of program                                        */
/*********************************************************/
```

Appendix B. How to Get the Samples Used in this Book

The files listed below are available using anonymous FTP.

Source Code

- simparm.c
- armtest.c

Compiled Code

- armtest.exe for Windows 95/NT
- armtest.sun for Solaris 2.4
- armtest.aix.414 for AIX versions up to 4.1.4
- armtest.aix for AIX versions later than 4.1.4

Declarations

- For PowerBuilder
- For Visual Basic

Please follow the instructions in the appropriate section below, to get the files that you need.

For Users Within the IBM TCP/IP Network

1. Connect to rsserver.itso.ral.ibm.com using FTP.
2. Specify a user ID of anonymous.
3. Enter your mail address in place of a password.

You will find the samples in the directory /pub/SG242124.

For Users Outside the IBM TCP/IP Network

Point your Web browser to:
ftp://www.redbooks.ibm.com/redbooks/SG242124/

Appendix C. Installing Reporter

We installed Reporter Version 2.0 on our RS/6000, so that we could use its reporting functions for ARM data.

Reporter comprises two components:

- Reporter Manager

  In Version 2.0, the Reporter Manager must run on AIX; it cannot run on Windows NT, Sun or HP-UX. However, since Reporter runs on the Tivoli Management Framework, once installed it can be controlled from any Managed Node within its TMR.

- Reporter Agents

  If you just want to report ARM data, you do not need to install any Reporter Agents - the ARM agents will provide data for the Reporter Manager.

C.1 Software Used in the ITSO Reporter Environment

We used an RS/6000 for our Reporter Manager machine, with the following products installed *in the order shown below*:

- AIX 4.2.0.0
- Tivoli Management Framework Version 3.2
- Sybase SQL Server V11
- AIX 4.2 bos.compat

  This installs the AIX 4.2 compatibility links. If these links are not installed, the Reporter installation shell script will not complete successfully.

- Reporter Version 2.0

C.2 Installing Reporter Components

The installation steps are documented in *Reporter Installation and Administration Version 2.0*, SH19-4119.

Before proceeding, we *strongly* recommend that you perform a backup of your Tivoli database. The procedure for doing this is documented in Appendix G, “Backing Up Your Tivoli Database” on page 145.

Installation is performed from the Tivoli Desktop using the Install Product function, and must be done in this order:

1. TME 10 Reporter 2.0 Desktop object

   The Desktop object must be installed on both the TMR server machine and the Reporter Manager machine. In our scenario, the TMR server and the Reporter Manager were on the same machine, so we only installed the Desktop object on that one machine.

2. TME 10 Reporter 2.0 Manager
3. TME 10 Reporter 2.0 Clients

   This component is not required for ARM.

4. TME 10 Reporter Sybase DataBase

   We used Sybase. If you use DB2 or Oracle, just substitute the appropriate Reporter component here.

After installation is completed, you should see a new icon on the TME Desktop, as shown in Figure 86.

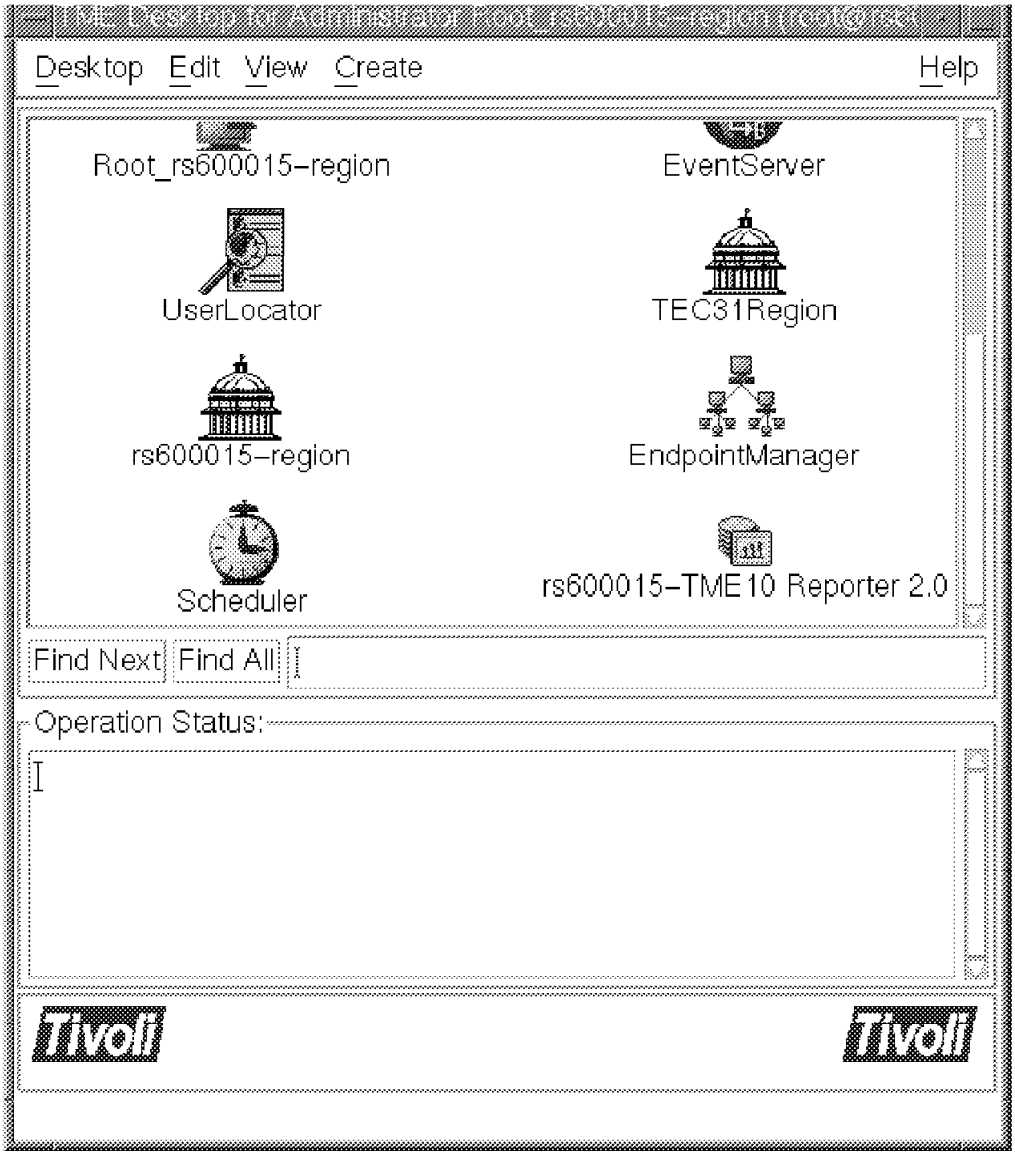


Figure 86. The Reporter Icon on the TME Desktop

C.3 Preliminary Setup

We added the following line to the file `/usr/lpp/perfrep/bin/prsetenv`:

```
su - perfrep
```

This would enable us to start Reporter from the Desktop icon, once we had completed the rest of the definitions in this appendix.

Then we created an AIX user ID called *perfrep*, in the *adm* group.

C.4 Setting up Reporter

We used a Sybase database for Reporter, and the following instructions are therefore Sybase-specific. They do not necessarily apply if you use Oracle or DB2.

We used the manual *Reporter Installation and Administration Version 2.0*, SH19-4119 to guide us through the Sybase setup. But we found a few gaps in the manual, and some of the required files were missing from the Reporter CD that we used. If you use this appendix in conjunction with the manual, and create the files as we have shown them, your installation should work. However, this appendix is not a complete replacement for the manual.

C.4.1 Setting Up the DBMS

We found that the file `/etc/perfrep/prsybaseprofile` was missing, so we created it in the directory `/home/perfrep/etc/` (so that it was under the home directory of our user ID) with the following contents:

```
SYBASE=/usr/local/sybase
export SYBASE

PATH=${PATH}:$SYBASE/bin
export PATH

DSQUERY=SYBASE
export DSQUERY
```

This file is used to set and export the DBMS environment variables for the database users.

The next step was to customize the perfrep.cfg file.

We copied the default file from the directory `etc/perfrep/` to the directory `/home/perfrep/etc/` so that it was underneath the home directory of our user ID.

Then we customized it to match our environment. You can see the changes we made highlighted in the file below.

```
#*************************************************************************
# Section for database parameters.
#*************************************************************************
PR_Db ZCC_Db
# The database management system (DB2, ORACLE or SYBASE).
dbms      = SYBASE
# The table prefix.
tabPrefix = PERFREP
# The table prefix must be in UPPERCASE
# The default table space to install additional data tables in.
tabSpace  = USERS
# The database user ID or group (role) of the Performance Reporter
# user.
userGroup = pruser
# The database user ID or group (role) of the Performance Reporter
# administrator.
adminUser = pradm
# The file which contain necessary environment database setups.
dbSetup   = /home/perfrep/etc/prsybaseprofile
# The database user and password
user = perfrep
password = garypw
#include $HOME/.zccdb
#*************************************************************************
```

Next we edited the .profile file for our AIX user ID *perfrep*. You can see that we set the environment variable ZCCPARM to our perfrep.cfg file and that we set our *prsybaseprofile* to execute.

```
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:$HOME/bin:/usr/bin/X11:/sbin:.
export PATH

ZCCPARM=/home/perfrep/etc/perfrep.cfg
export ZCCPARM

. /home/perfrep/etc/prsybaseprofile
```

We decided to call our Reporter database *reporter*.

Then, after logging on with the *perfrep* user ID, we issued the following command at the AIX prompt to invoke the Sybase customization facility:

```
isql -Usa -P
```

This enabled us to run the Sybase customization commands that you can see below.

```
1> drop database reporter
2> go
Msg 3701, Level 11, State 6:
Line 1:
Cannot drop the database 'reporter', because it doesn't exist in the system
catalogs.
1> create database reporter on default = 20
2> go
CREATE DATABASE: allocating 10240 pages on disk 'master'
1> sp_dboption reporter, "allow nulls by default", true
2> go
Database option 'allow nulls by default' turned ON for database 'reporter'.
Run the CHECKPOINT command in the database that was changed.
(return status = 0)
1> sp_dboption reporter, "ddl in tran", true
2> go
Database option 'ddl in tran' turned ON for database 'reporter'.
Run the CHECKPOINT command in the database that was changed.
(return status = 0)
1> use reporter
2> go
1> CHECKPOINT
2>go
1>quit
```

Note that we set the size of the database to 20 MB. You should use the manual *Reporter Installation and Administration Version 2.0*, SH19-4119 to calculate the size of the database that you will require, since this will depend on the number of Reporter agents and components that you will use.

C.4.2 Defining the Groups, Administrators and Users

Reporter allows two classes of users:

- Administrator can set up and control Reporter, and run reports.
- User can run reports.

Initially, we intended just to create an administrator, but we found that if the group for users, *pruser*, is not created, the prsetupdb script will fail. So we also created *pruser*.

To do this, we logged on to AIX with the *perfrep* user ID that we had created in C.3, “Preliminary Setup” on page 124 and issued the following command at the AIX prompt to invoke the Sybase customization facility:

```
isql -Usa -P
```

This enabled us to run the Sybase customization commands that you can see below.

Here the *use reporter* defines our database, and names it *reporter*. Actually any name could have been used.

The line

```
1> sp_adduser perfrep, PERFREP, pradm
```

might look a bit strange. Here *perfrep* is the user ID and *PERFREP* is the table prefix, which must match the perfrep.cfg file.

```
1> use reporter
2> go
1> sp_addgroup pradm
2> go
New group added.
(return status = 0)
1> grant create table, create view, create default to pradm
2> go
1> grant create procedure, create rule to pradm
2> go
1> sp_addgroup pruser
2> go
New group added.
(return status = 0)
1> grant create procedure to pruser
2> go
1> sp_addlogin perfrep, garypw, reporter
2> go
1> sp_adduser perfrep, PERFREP, pradm
2> go
New user added.
(return status = 0)
1> quit
```

At this point you will have set up the files as shown in Figure 87 on page 129.

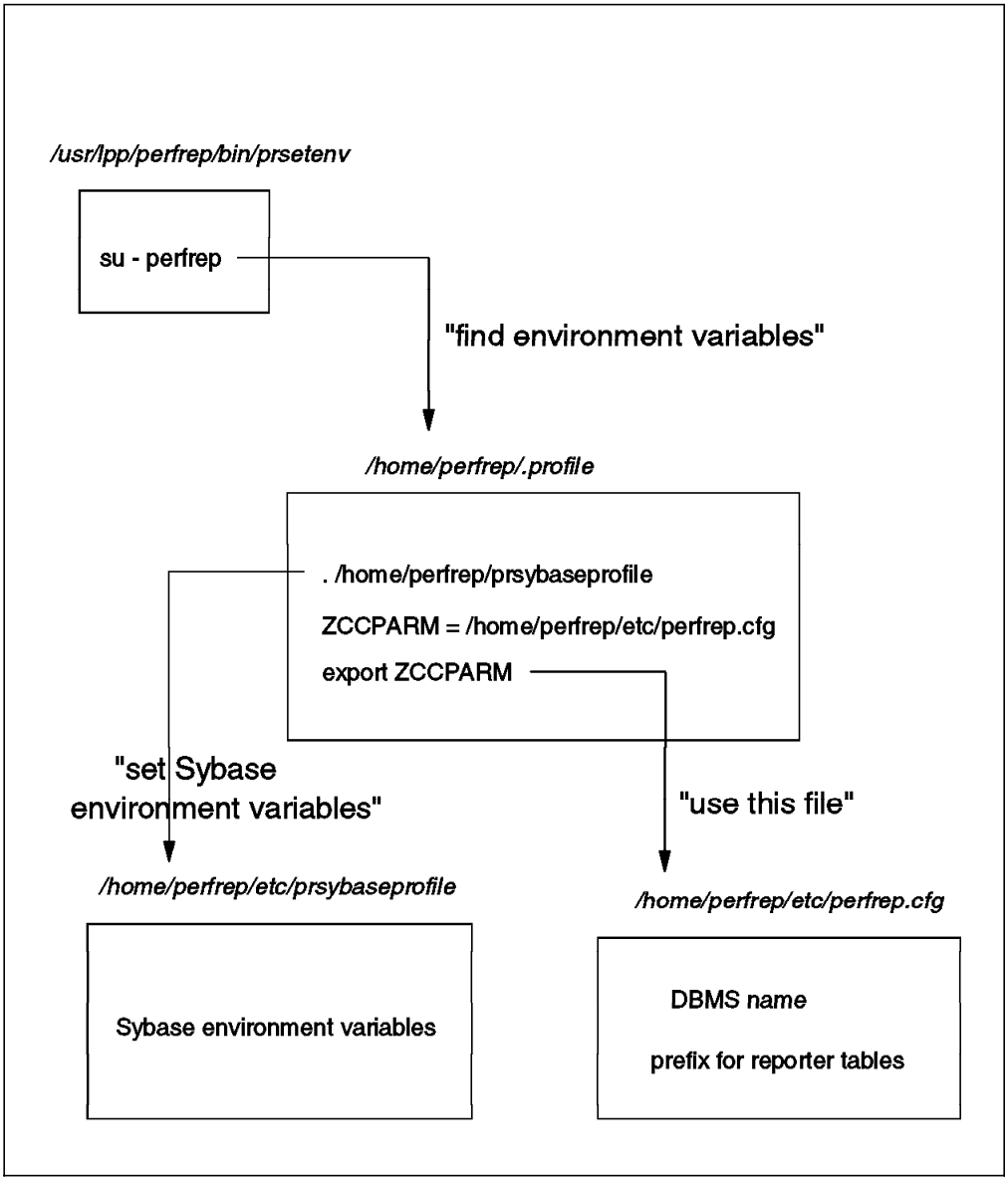


Figure 87. The Links Between the Customized Files

C.4.3 Creating the Reporter System Tables

The script *prsetupdb* requires a file called *ctlib.loc*. This file was not present in our Sybase directory, causing the script to fail.

The file is part of the Sybase Client. We obtained a copy of the file, and installed it in the directory */usr/local/sybase/locales/us_english/iso_1/*.

Then we changed the owner and group for the file by issuing the following commands:

```
chown perfrep ctlib.loc
chgrp adm ctlib.loc
```

We checked that the file permissions included executable. If they do not, change them by issuing the command:

```
chmod 777 ctlib.loc
```

Now issue the whoami command, to make sure that you are logged in as *perfrep*, and then issue the command echo $SYBASE, to ensure that you have the environment variables set; they will not be set if you used su to switch user to *perfrep*.

Now run the prsetupdb script.

The prsetupdb script should execute from the /usr/bin/ directory, using a link to the /usr/lpp/perfrep/bin/prsetupdb directory, where the script is installed.

You will be asked to confirm the parameters for the system tables as you can see below.

```
This script verifies that your database management system
and database user are
set up correctly, and creates the Performance Reporter system tables.

PRINS016I  The following values will be used:
PRINS011I  DBMS.................: SYBASE
PRINS013I  Table prefix.........: PERFREP
PRINS012I  Table space..........: USERS
PRINS014I  User group...........: pruser
PRINS015I  Administrator group...: pradm

PRINS017Q  Press Enter to continue, or type any nonblank character to mod
 values.
```

Here, you will see that the table prefix is set to *PERFREP*, whereas the manual *Reporter Installation and Administration Version 2.0*, SH19-4119, states that it should be *perfrep*, the same as the administrator user ID. The manual is wrong.

Next you should see messages indicating that the setup of each part completed successfully.

```
PRINS009I  prsetupdb: Messages will be spooled to the log file /home/perfrep/prsetupdb.log

PRINS002I  Creating Reporter system tables, using PERFREP as table
prefix.

PRINS010I  Initiating Reporter system tables for PERFREP.

PRINS028I  Inserting report templates.

PRINS026I  Inserting calendar data.

PRINS027I  The database is successfully set up.

PRADM017Q  Press Enter to exit, or type any nonblank character to display
           the log file /home/perfrep/prsetupdb.log
```

Note: If you get the error message:

```
PRCOM010E Prsetupdb: zccdbcmd did not execute the SQL state-
ment. Select in Sybase, The return code is '2'.
```

there may be a problem with the ctlib.loc file.

Note: If you get an error message indicating a problem with the Sybase password, check the perfrep.cfg file. The line:

```
#include $HOME/.zccdb
```

should have a # at the beginning. This would comment it, which is what you want.

C.5 Preparing to Install Reporter Components

We had had some problems with the tempdb database being filled up with the Sybase transaction log, so we issued the following commands to fix it.

```
$ isql -Usa -P

1> sp_dboption tempdb, "trunc log on chkpt", true
2> go
Database option 'trunc log on chkpt' turned ON for database 'tempdb'.
Run the CHECKPOINT command in the database that was changed.
(return status = 0)
1> use tempdb
2> go
1> CHECKPOINT
2> go
1> quit
```

C.6 Installing the Sample Component Shipped with Reporter

We used this component to verify our installation. When installed, it puts data into the Reporter database and provides a set of reports to display that data.

We followed the Reporter manual, to perform the installation.

We typed `smit perfrep` to start Reporter SMIT, then chose **Install and configure**, followed by **Install components**, then **Install component**.

At that point we saw a screen similar to the following:

```
                              Install component

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Component name                                     []
* Table space name                                   [USERS]
  Table prefix                                        PERFREP
* Administrator                                       pradm
* List of users                                      [pruser]
```

Press PF4 to see a list of the components that can be installed. Select **Sample** and press Enter.

You should see messages similar to the following:

```
Checking the component Sample, please wait.

Installing the component Sample, please wait.

prinstcomp: The component Sample is successfully installed.

Press Enter to exit, or type any nonblank character to display the log
file /var/perfrep/prinstsample.log
```

At this point, you can create and display the sample reports, following the procedure in the Reporter manual.

You can see our sample in Figure 88.

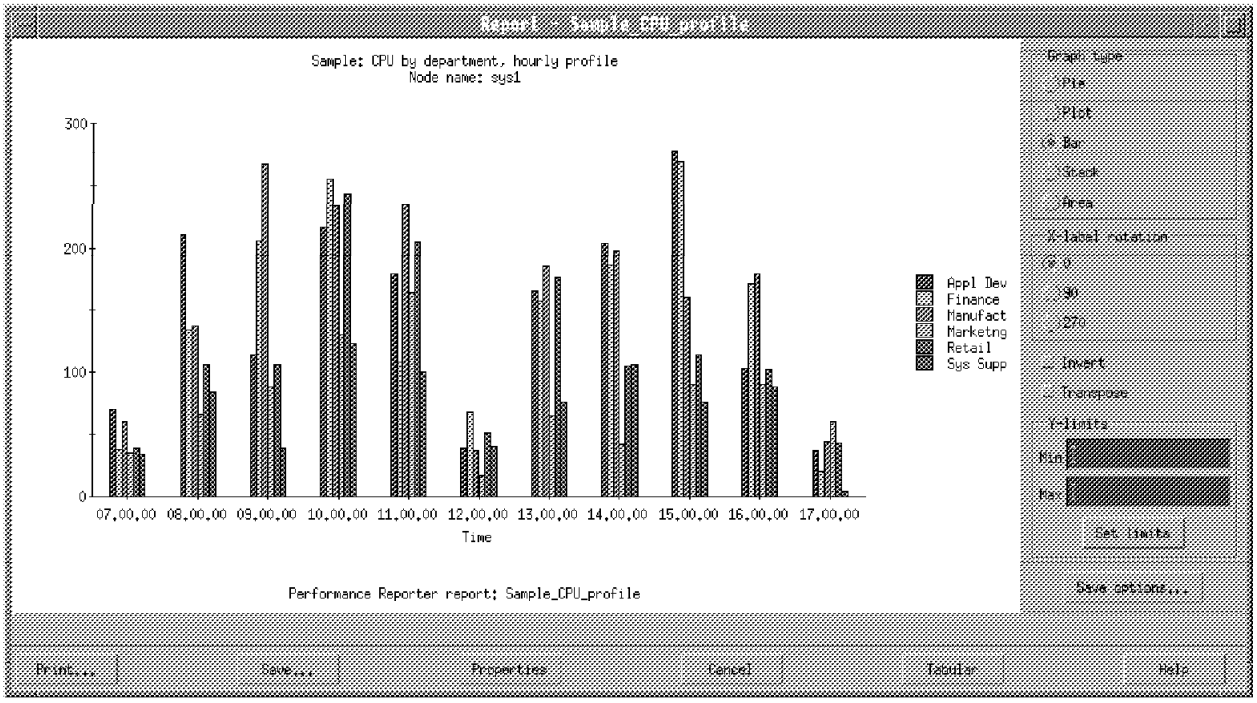


Figure 88. Reporter Sample Report

Appendix D. The Tivoli Desktop for Windows

The Desktop for Windows is a very useful application provided with the Tivoli Framework.

It enables you to interface with the TME server without using an X-Windows package, or a X-station. The Desktop looks exactly like what you would see on the console of the TMR server.

Why would you want to install this rather than a remote X-based server, or an X-Windows application?

- Desktop for Windows performs better than a remote x-based connection server.
- You can drag and drop between the Windows Desktop and the Tivoli Desktop.
- It encodes the logon password, providing better security
- You use the Desktop for any TMR, and for more than one TMR at the same time. You do not actually need to be a part of the TMR.

D.1 Installing the Desktop

The Desktop for Windows code is provided on the Tivoli Framework CD.

Installation is quick and simple for Windows 3.1, Windows 95, and Windows NT. It is a little more complex for OS/2.

D.1.1 Installing on Windows

To install the Desktop, insert the Tivoli Framework CD in your CD drive, and run the setup program appropriate for your version of Windows.

- For Windows 95 and Windows NT, type `x:\pc\Desktop\disk1\setup.exe`
- For Windows 3.1, type `x:\pc\win32s\disk1\setup.exe`

Where x is the drive letter of your CD-ROM.

D.1.2 Installing on OS/2

Installation on OS/2 requires running a script to install the Win-OS/2 package (if not already installed). After this preparation, win32s Version 1.25a can be installed.

Complete instructions for OS/2 are provided in the *TME 10 Desktop for Windows User's Guide*.

Appendix E. Setting the Tivoli Environment Variables for a DOS Window on NT

If you want to issue warmcmd commands from a DOS window, on a Windows NT server as we did, you will need to set the Tivoli environment variables each time you open a DOS window.

We found this rather tedious, so we created a shortcut to the DOS window and customized this to set up the Tivoli environment variables each time it was started.

Note: This will only work from the shortcut; it cannot be set up for the DOS window opened from the Start menu.

Follow this procedure to create the shortcut.

1. Start Windows Explorer.
2. Select the **winntprofilesAdministratorStart MenuPrograms** directory.
3. Within the directory, select the **Command Prompt** shortcut, and copy it to the Windows Desktop.
4. Right click this shortcut on the Windows Desktop, and select **Properties**.
5. On the resulting menu, select the **Shortcut** tab.
6. Place the cursor at the end of the text in the Target box.
7. Add the following string:

   ```
   /k winntsystem32driversetcTivolisetup_env.cmd
   ```

8. Click on **OK**.

Now when you open a DOS window by double-clicking the **Command Prompt** icon on the Desktop, you should see the following message in the window:

```
Tivoli environment variables configured.
```

Appendix F. Granting Permissions for the Tivolidb Directory on NT

This procedure is performed on the TMR server using an account with Administrator authority to this directory:

1. From Windows Explorer, right click on the **\Tivoli\db** subdirectory.
2. From the resulting menu, select the **Properties** option.
3. Now select the **Security** tab, and then click on **Permissions**, as shown in Figure 89.

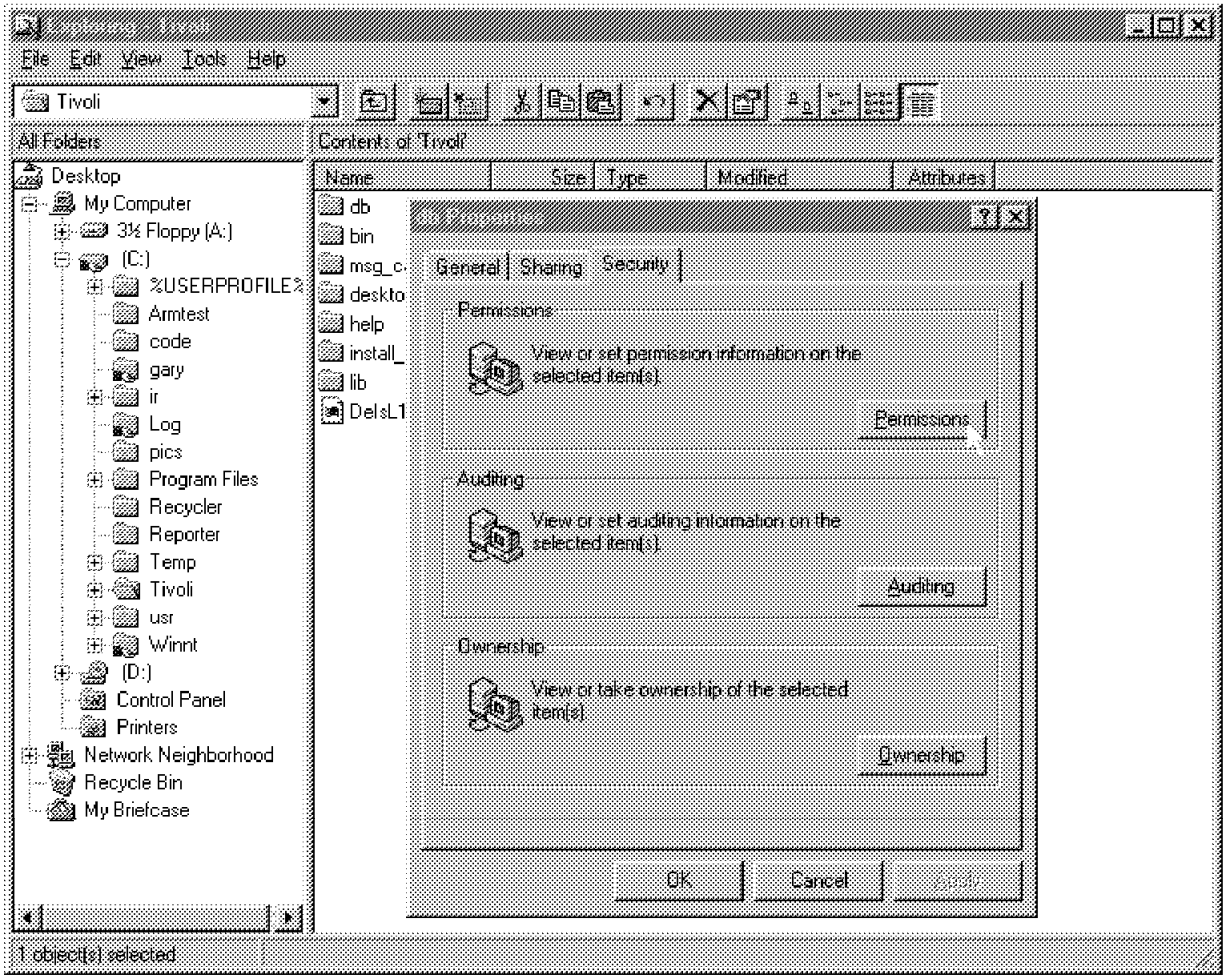


Figure 89. Opening the Directory Permissions Dialog Box

4. In the resulting dialog box, click on **Add**, as shown in Figure 90.

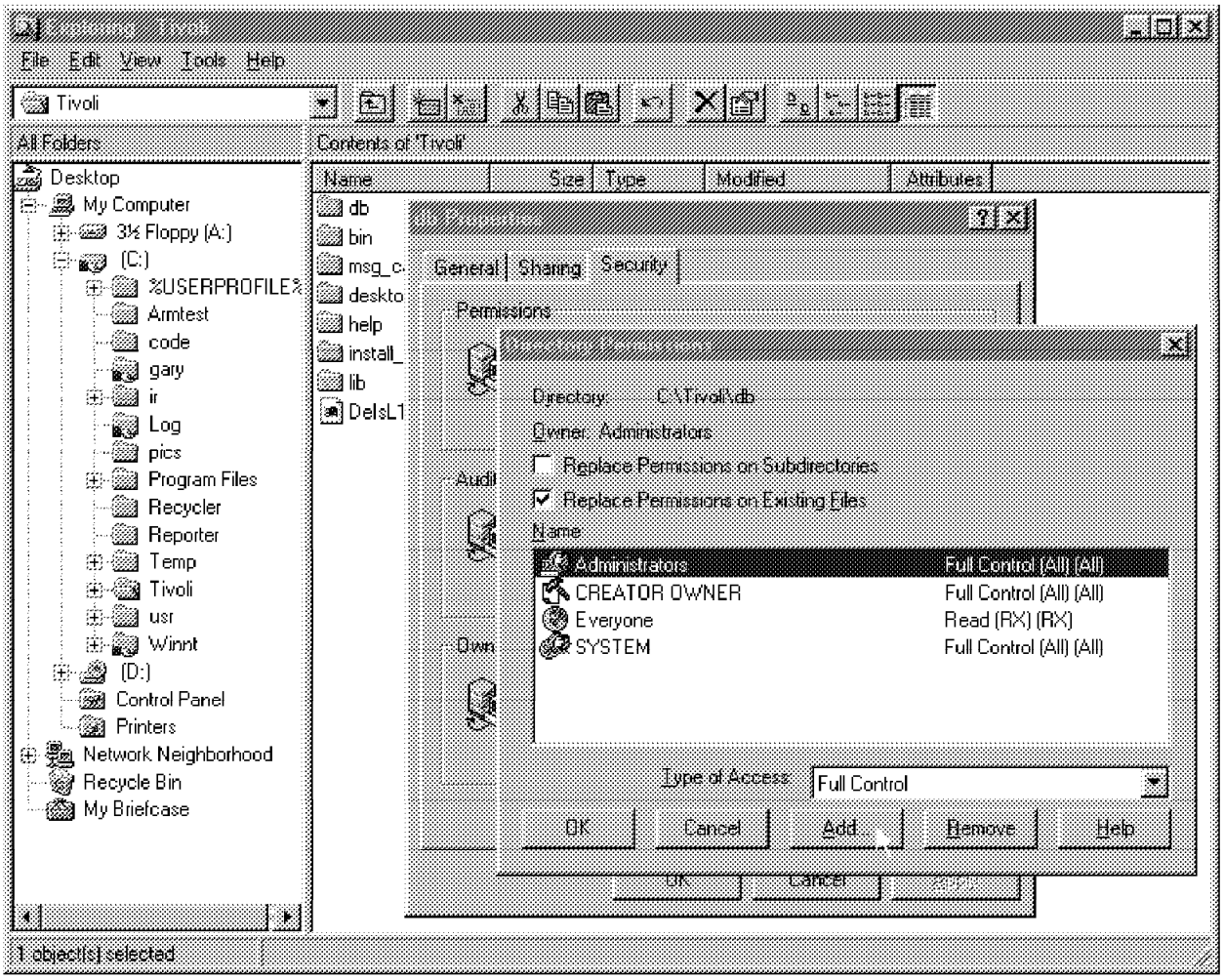


Figure 90. The Directory Permissions Dialog Box

On the resulting Add Users and Groups dialog box, select **Show Users**.

Now you will see a list of all the available users, in the Names pane.

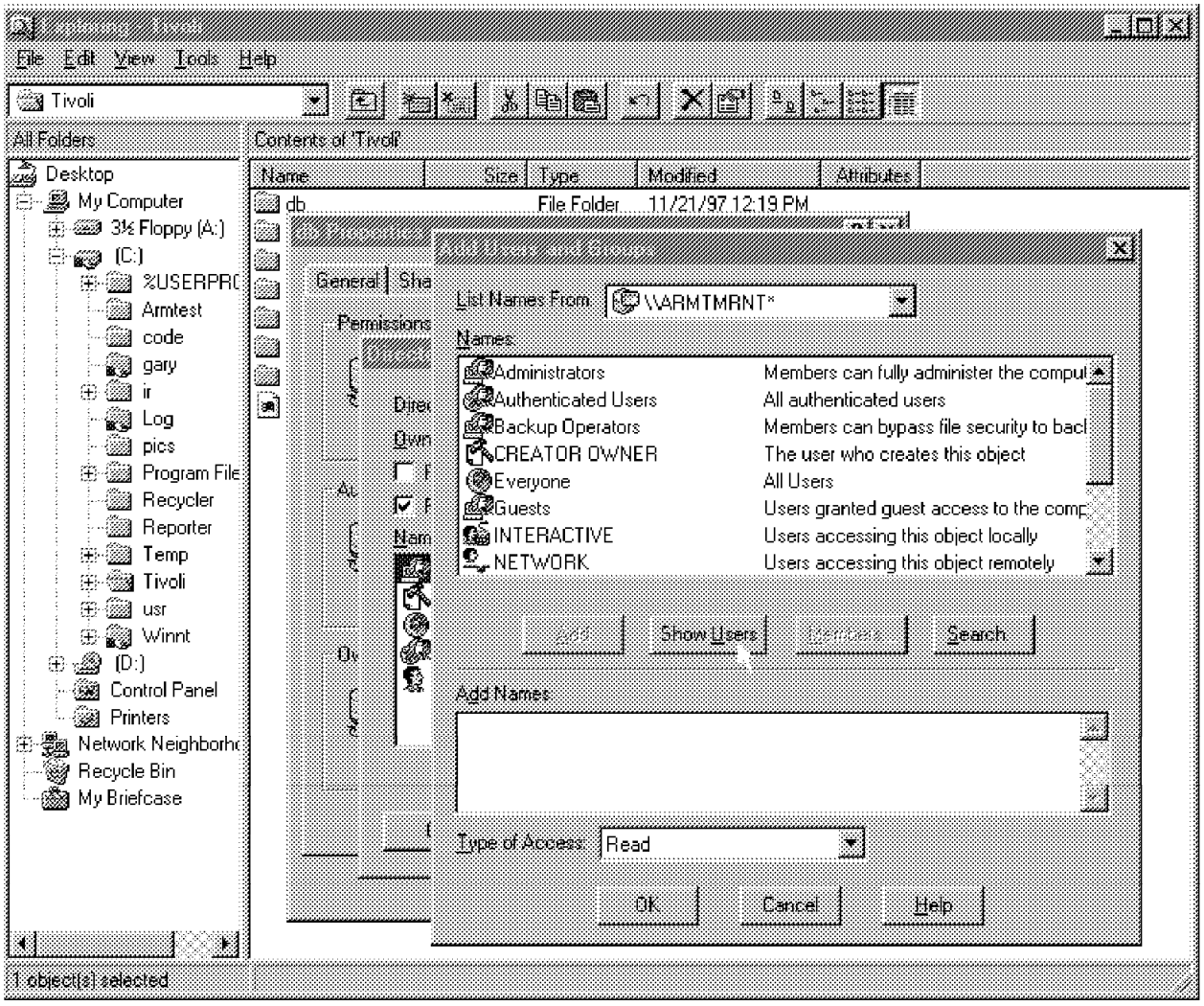


Figure 91. The List of Users

5. From the list of names, select **tmersrvd(tmersrvd)**, then click on **Add**, as shown in Figure 92.

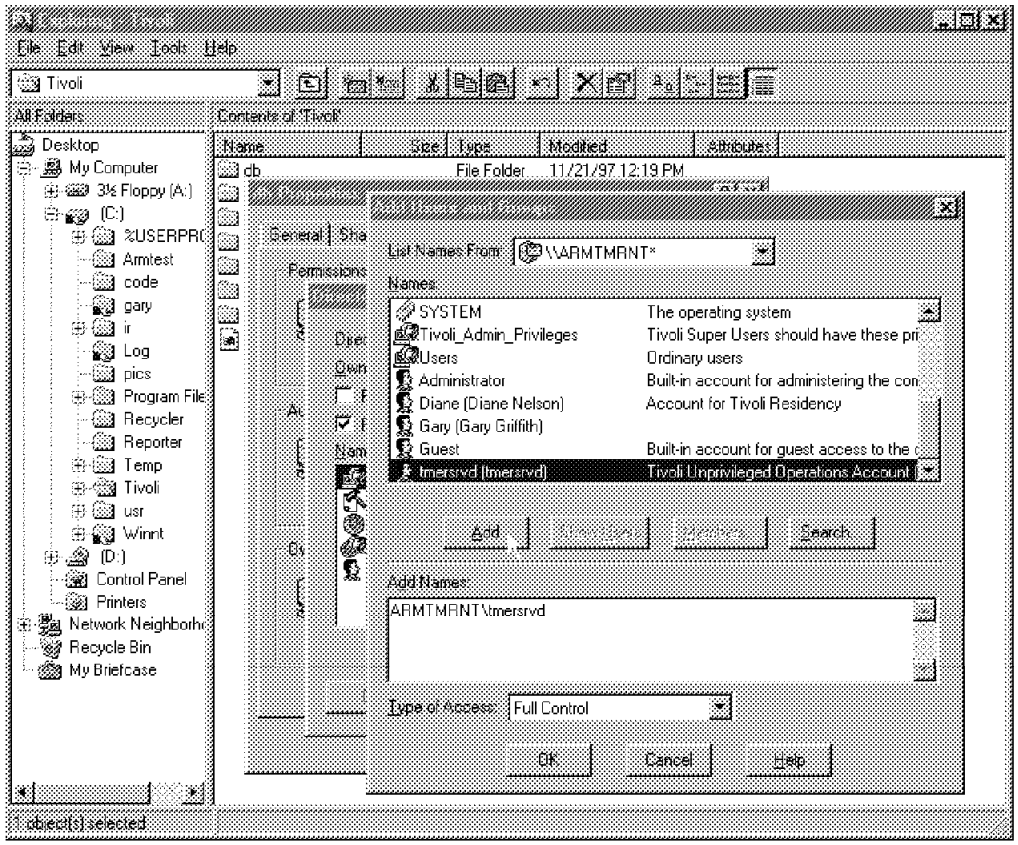


Figure 92. Adding the tmersrvd User

6. Change the Type of Access option to Full Control.

   Now the name of the TMR server will appear in the Add Names pane. Then click on **OK**, as shown in Figure 93.

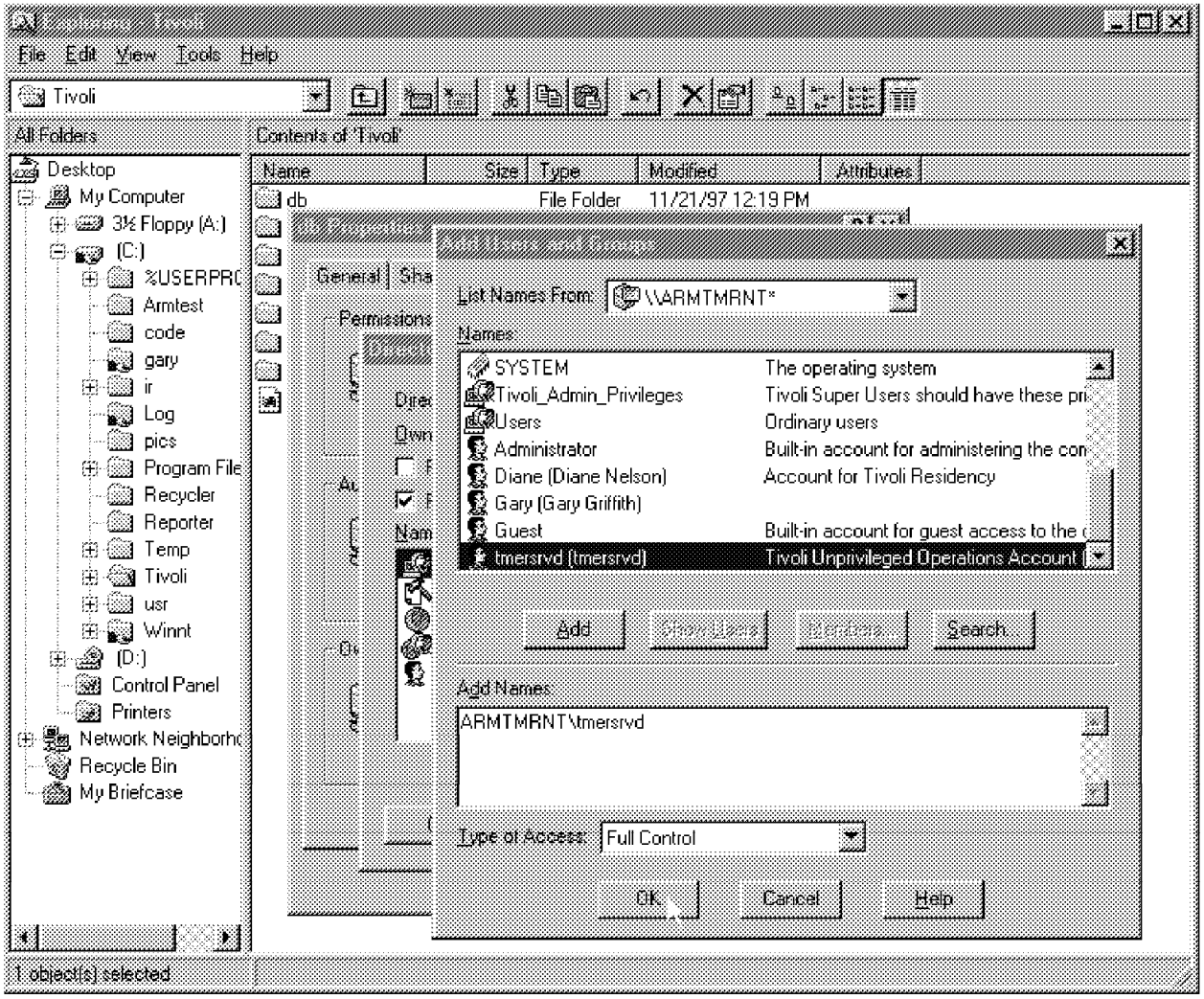


Figure 93. Granting the tmersrvd User Full Control

Now you will be back at the Directory Permissions dialog box.

7. Check the **Replace Permissions on Subdirectories** box, and then click on **OK**, as shown in Figure 94.

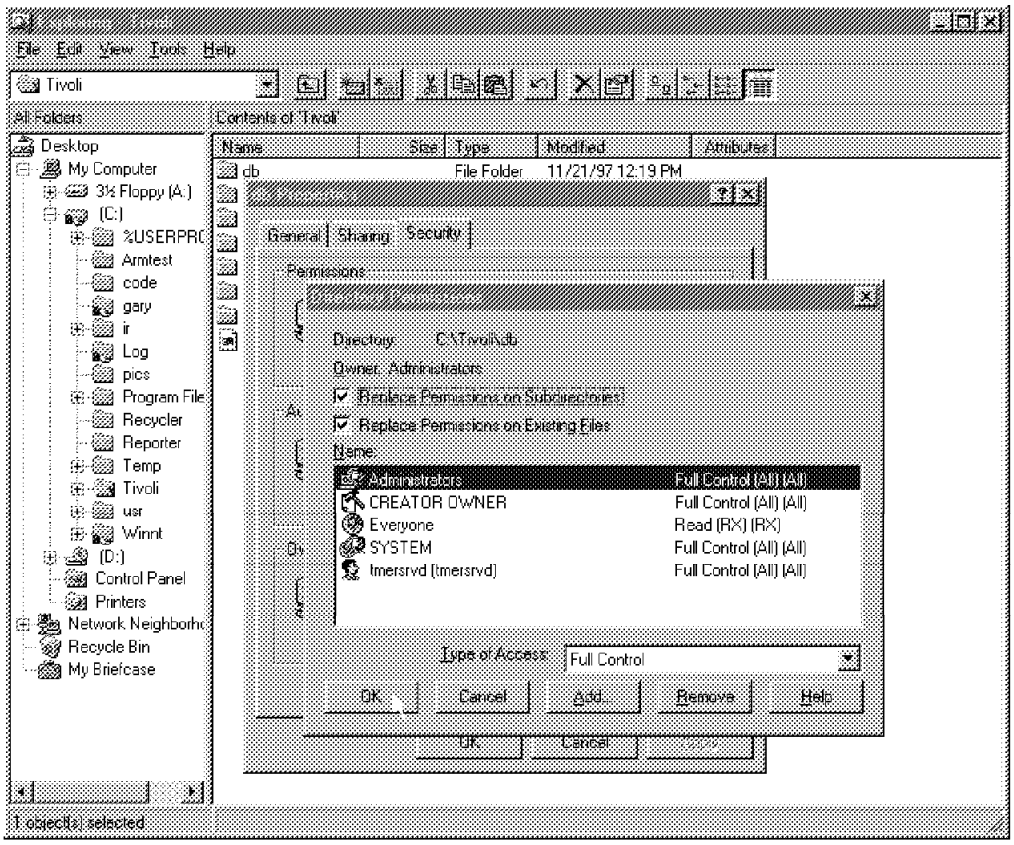


Figure 94. Confirming the Directory Permissions

8. Click on **Yes** in response to the Windows NT message asking if you want to replace the security information, as shown in Figure 95.

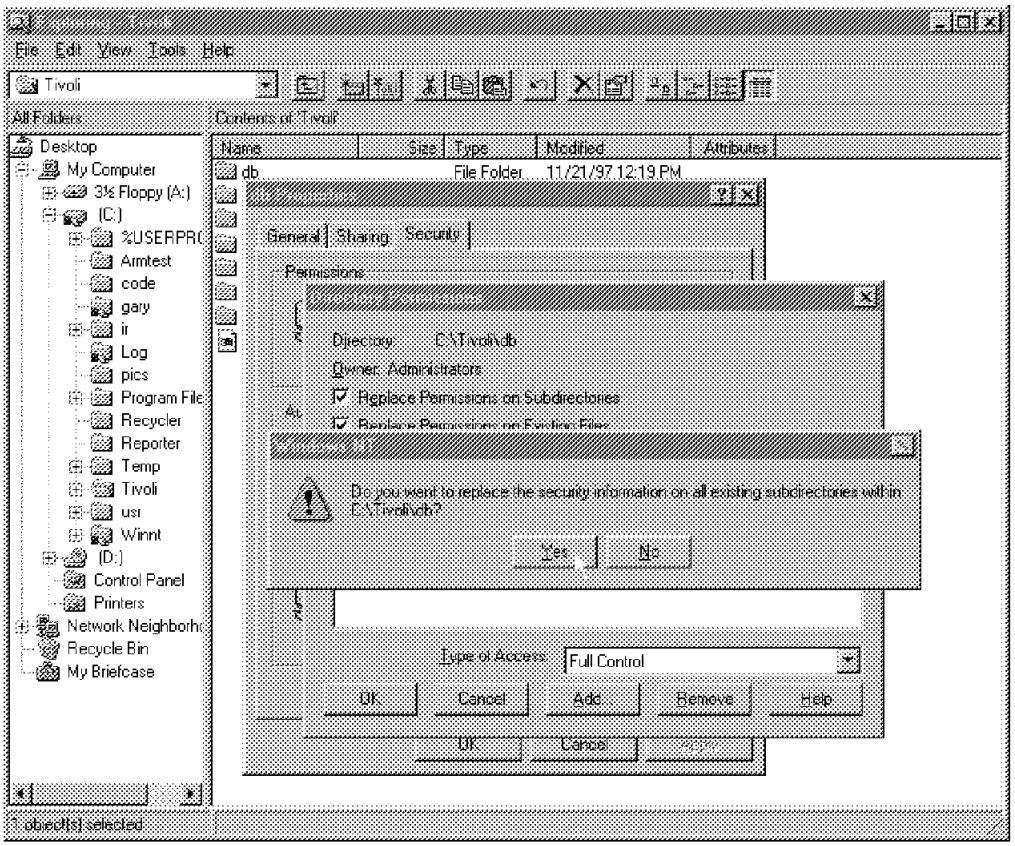


Figure 95. Replacing the Security Information

9. Now click on **OK** to close the db Properties dialog box.

Appendix G. Backing Up Your Tivoli Database

Before and after each application installation, Tivoli recommends that you back up your Tivoli database. This makes it possible to go back to the pre-application database if you encounter a problem while installing a particular product.

G.1.1 Creating a Backup

Complete the following steps to make a backup of the Tivoli database.

1. Log in as root or as any other TME administrator with the super or backup role.
2. Click on **Desktop** on the Desktop menu bar and select **Backup** from the pull-down menu.

   This will display the Backup Tivoli Management dialog box.
3. Double-click on the managed node in the Available managed node scrolling list, and it will be moved to the Backup these managed nodes scrolling list.
4. Click on the **Start Backup** button.

An image of your database will be created in the default directory: /var/spool/Tivoli/backups/DB_%t.

For more information, refer to the *TME 10 Framework Reference Manual*.

G.1.2 Restoring a Backup

You can restore the backup using the wbkupdb command. You will need the senior or restore authorization role to do this.

For example:

```
wbkupdb -r -d /var/spool/Tivoli/backups/DB_Mar19-1400 rs600015
```

Where:

-r Restores the database for the specified node

-d Specifies the name of the backup file

The restore function cannot be performed from the Desktop. It must be run from the command line.

For more information refer to the *TME 10 Framework Reference Manual*.

G.2 Framework Reinstallation

It is possible that you could find yourself in a situation where IBM Support suggests you reinstall the Framework. These steps can be used to accomplish this:

- The oserv process and other product processes should be shut down.
- Delete all the files in /usr/local/Tivoli directory.
- Delete all files in the /var/spool/Tivoli directory.

- Mount the Framework CD-ROM at the /dev/cd0 as /cdrom directory.
- Create the directory install_dir in the /usr/local/Tivoli directory.
- Change directory to /usr/local/Tivoli/install_dir.
- Issue the `/cdrom/wpreinst.sh` command.
- Issue the `/wserver -c /cdrom` command.
- Set the initial parameters as license key and type of encryption.

For more information, refer to the *TME 10 Framework Planning and Installation Guide*.

Appendix H. Special Notices

This publication is intended to help applications analysts to monitor the performance, usage and availability of the application programs for which they are responsible. The information in this publication is not intended as the specification of any programming interfaces that are provided by TME 10 Distributed Monitoring. See the PUBLICATIONS section of the IBM Programming Announcement for TME 10 Distributed Monitoring for more information about what publications are considered to be product documentation.

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

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

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

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

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

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (″vendor″) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

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.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other

operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

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
OS/2
RS/6000

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and all Java-based trademarks and logos are trademarks or registere trademarks of Sun Microsystems, Inc. in the United States and other coun

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.

ActionMedia, LANDesk, MMX, Pentium, and ProShare 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.

Tivoli, TME and TME10 are trademarks of Tivoli Systems Inc. in the United States

Lotus Notes is a trademark of Lotus Development Corporation.

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

Appendix I. Related Publications

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

I.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 151.

- *A First Look at TME 10 Distributed Monitoring 3.5*, SG24-2112
- *Measuring Lotus Notes Response Times with Tivoli's ARM Agents*, SG24-4787
- *Migrating from Systems Monitor for AIX to TME 10 Distributed Monitoring*, SG24-4936
- *TME 10 Cookbook for AIX Systems Management and Networking Applications*, SG24-4867

I.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

I.3 Other Publications

These publications are also relevant as further information sources:

- *Application Response Measurement API Guide second edition*
- *Distributed Monitoring Collection Reference, Version 3.5*, SC31-5118
- *Distributed Monitoring Users Guide, Version 3.5*, GC31-8382-01
- *TME 10 Reporter Installation and Administration, Version 2*, SH19-4119-02

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at http://www.redbooks.ibm.com/.

How IBM Employees Can Get ITSO Redbooks

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

- **Redbooks Web Site on the World Wide Web**

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

- **PUBORDER** — to order hardcopies in the United States
- **Tools Disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLCAT REDPRINT
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type the following command:

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

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1998
  ```

- **REDBOOKS Category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL

Redpieces

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

How Customers Can Get ITSO Redbooks

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

- **Online Orders** — send orders to:

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

- **Telephone Orders**

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

- **Mail Orders** — send orders to:

IBM Publications 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

- **On the World Wide Web**

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

Redpieces

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

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

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

List of Abbreviations

API	Application Programming Interface
ARM	Application Response Measurement
DM	Distributed Monitoring
IBM	International Business Machines Corporation
ITSO	International Technical Support Organization
SIS	Software Installation Services
TEC	Tivoli Enterprise Console
TME	Tivoli Management Environment
TMP	Tivoli Management Platform
TMR	Tivoli Management Region

Index

Numerics

A

B

C

D

F

G

H

I

L

M

P

R

S

T

U

V

W

ITSO Redbook Evaluation

Using Tivoli's ARM Response Time Agents
SG24-2124-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
__Customer __Business Partner __Solution Developer __IBM employee
__None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

_____

_____

_____

_____

_____

SG24-2124-00
Printed in the U.S.A.