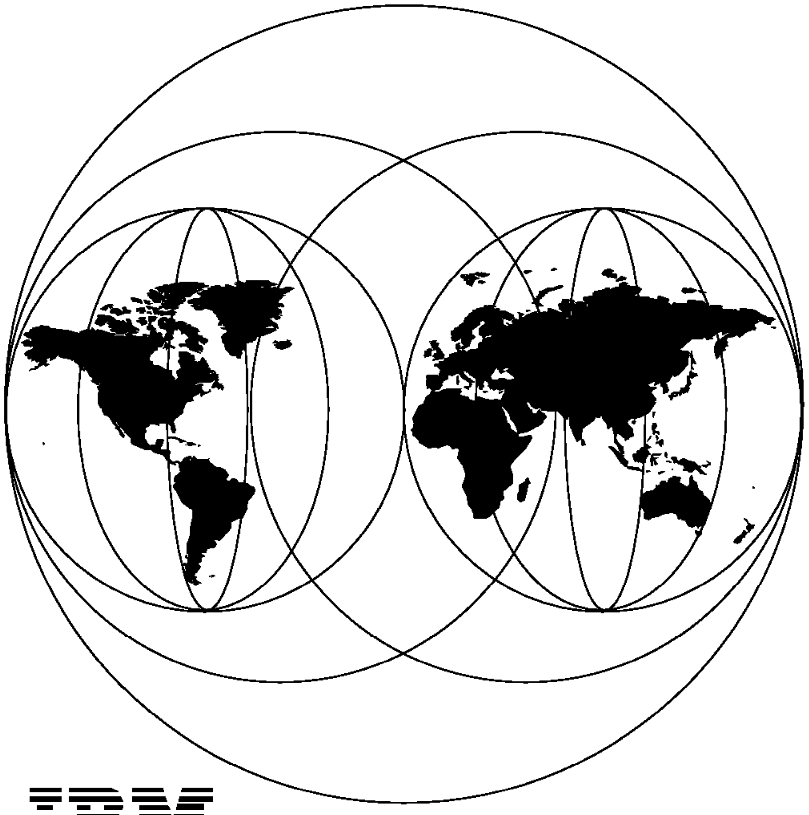
International Technical Support Organization

SG24-4515-00

Examples Using NetView for AIX Version 4

December 1995



IBM

International Technical Support Organization
Raleigh Center

International Technical Support Organization

SG24-4515-00

Examples Using NetView for AIX Version 4

December 1995

Take Note!

Before using this information and the product it supports, be sure to read the general information under "Special Notices" on page xi.

First Edition (December 1995)

This edition applies to Version 4 Release 1 of the NetView for AIX feature of of IBM SystemView for AIX, Program Number 5765-527 for use with RISC System/6000 AIX.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader′s feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. HZ8D Building 678
P.O. Box 12195
Research Triangle Park, NC 27709-2195

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

Abstract

This document is intended to provide the network and systems management professional with details of the new features in NetView for AIX Version 4. It illustrates these features by means of practical examples of how the functions in NetView for AIX Version 4 can be used to address network and system management challenges.

This document is intended to supplement the standard product documentation regarding NetView for AIX and its related family of products. Although the focus of this document is on an individual feature of SystemView for AIX, it is intended to contribute to the growing body of information that shows how NetView for AIX participates in a SystemView strategy.

This document is intended for personnel who will be administrating or customizing NetView for AIX. A general knowledge of NetView for AIX and TCP/IP network management is assumed. Some of the examples make use of C language programming, and the reader may need some knowledge of programming in an AIX environment to fully benefit from them.

(285 pages)

Contents

Figures

Tables

Special Notices

Preface

This document is intended to provide the network and systems management professional with details of the new features in NetView for AIX Version 4. It illustrates these features by means of practical examples of how the functions in NetView for AIX Version 4 can be used to to address network and system management challenges.

This document is intended to supplement the standard product documentation regarding NetView for AIX and its related family of products. Although the focus of this document is on an individual feature of SystemView for AIX, it is intended to contribute to the growing body of information that shows how NetView for AIX participates in a SystemView strategy.

How This Document is Organized

The document is organized as follows:

- Chapter 1, “Introduction”

  This chapter provides overall information regarding the project involved in creating this document and summarizes the enhancements to NetView for AIX in Version 4.

- Chapter 2, “Client/Server Support in NetView for AIX”

  Describes the elements of the Client/Server feature of NetView for AIX Version 4 and shows some examples of the performance impact of it. It also discusses aspects of converting a program that uses the NetView for AIX APIs to the Client/Server environment.

- Chapter 3, “Security”

  Describes the security feature of NetView for AIX Version 4 and shows examples of how to implement and customize it.

- Chapter 4, “The Collection Facility”

  Describes how to use the Collection Facility to organize network resources into logical groups. It also shows examples of use of the Collection Facility APIs.

- Chapter 5, “Introducing NetView for AIX Event Rulesets,” Chapter 6, “Examples of NetView for AIX Rulesets” and Chapter 7, “Using the Collection Facility with Event Rulesets”

  These chapters describe what Event Rulesets are, how they are invoked and then provide a number of practical ruleset examples.

- Chapter 9, “Agent Policy Manager (APM)”

  Summarizes the capabilities of the Agent Policy Manager feature by means of worked examples.

- Chapter 10, “NetView for AIX Open Topology”

  Describes the Open Topology API and illustrates its use with a sample program.

- The appendices includes instructions on how to obtain the code samples from this book and listings of some of the sample programs.

Related Publications

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

- *NetView for AIX Administrator's Guide, Version 4*, SC31-8168-00
- *NetView for AIX Administrator's Reference, Version 4*, SC31-8169-00
- *NetView for AIX Diagnosis Guide, Version 4*, SC31-8162
- *NetView for AIX Programmer's Guide, Version 4*, SC31-8164
- *NetView for AIX Programmer's Reference, Version 4*, SC31-8165
- *NetView for AIX User's Guide for Beginners, Version 4*, SC31-8158-00

All of these publications are also provided in Dynatext (browsable) format if you install the nv6000.books component of NetView for AIX.

International Technical Support Organization Publications

- *Examples of Using Netview for AIX (V3)*, GG24-4327
- *IBM Systems Monitor Anatomy of a Smart Agent*, GG24-4398

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

To get a catalog of ITSO redbooks, VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

A listing of all redbooks, sorted by category, may also be found on MKTTOOLS as ITSOCAT TXT. This package is updated monthly.

How to Order ITSO Redbooks

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-445-9269. Almost all major credit cards are accepted. Outside the USA, customers should contact their local IBM office.

Customers may order hardcopy ITSO books individually or in customized sets, called BOFs, which relate to specific functions of interest. IBM employees and customers may also order ITSO books in online format on CD-ROM collections, which contain redbooks on a variety of products.

ITSO Redbooks on the World Wide Web (WWW)

Internet users may find information about redbooks on the ITSO World Wide Web home page. To access the ITSO Web pages, point your Web browser to the following URL:

```
http://www.redbooks.ibm.com/redbooks
```

IBM employees may access LIST3820s of redbooks as well. Point your web browser to the IBM Redbooks home page:

```
http://w3.itsc.pok.ibm.com/redbooks/redbooks.html
```

Acknowledgments

The authors of this document are:

Massimo Carnevali	IBM Italy
Luigi Casagrande	IBM Italy
Randy Craig	IBM US (NSD Raleigh)
Paul Fearn	IBM UK
Cristiane Petrillo Hilkner	IBM Brazil (Sumare)
Mark Hodge	IBM UK
Sergio Costa Lage	IBM Brazil (Sumare)
Blaz Mertelj	IBM Slovenia

This advisors for this document are:

Dave Shogren and Rob Macgregor
International Technical Support Organization, Raleigh Center

This publication is the result of residencies conducted at the International Technical Support Organization, Raleigh Center.

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

Barry Nusbaum
David Boone
International Technical Support Organization, Raleigh Center

Martha Crisson
Ken Chambers
Judith Dietz
Heather Kreger
Terry Simpson
Lee Cook
Cristina Zabeu
Lynnie Tierney
Pam Buckoski
Rich Hall
Dave Legrys
John Harmer
and the entire NetView for AIX development group, IBM Raleigh

Many members of Information Development, IBM Raleigh

Request for Feedback

Readers of this document are encouraged to feed back any information or comments regarding *any* of the material in this document. Please send your comments to:

Dave Shogren or Rob Macgregor
ITSO-Raleigh
VNET: SHOGREN at WTSCPOK or MCGREGOR at WTSCPOK

or: IBM International Technical Support Organization
Dept. HZ8D/B678/D100
1001 Winstead Drive
Cary, N.C. 27513

INTERNET: shogren@vnet.ibm.com
robmacg@vnet.ibm.com

Chapter 1. Introduction

NetView for AIX is the Network Operations Management feature of SystemView for AIX. NetView for AIX is primarily an SNMP-based management application. However it also provides platform functions that are used by many other management applications. The platform facilities that NetView for AIX provides may be summarized as follows:

Network mapping	NetView for AIX provides an extensive set of APIs to allow other applications to use its topology display capabilities.
Menu registration	Applications can add functions to the NetView for AIX menus. Such applications do not necessarily also make use of the network mapping APIs (although many do).
Event handling	NetView for AIX provides a standardized facility for processing and displaying event messages, based on SNMP traps.
Non-IP topology	The open topology API simplifies the integration of other network topologies along with the IP views.
Communications	Applications can also gain access to SNMP and CMOT protocols using APIs provided by NetView for AIX.

In Version 4, many enhancements have been made to NetView for AIX, affecting almost every aspect of its capabilities and use. In this book we describe examples which illustrate or exploit some of these enhancements, developed during projects at the ITSO-Raleigh center. The time available precluded us from covering all of the new features in equal depth. In this chapter, therefore, we will briefly introduce the enhancements before going on describe the detailed examples.

1.1 What′s New in NetView for AIX Version 4?

The following sections summarize the new features in NetView for AIX Version 4.

1.1.1 General

NetView for AIX Version 4 runs on either AIX Version 3.2.5 or 4.1.3 (or higher). When running on a Version 4.1.3 system it is compatible with the Common Desktop Environment (CDE).

NetView for AIX Version 4 is fully National Language Support (NLS)- and double-byte character set (DBCS)-enabled. It is available translated into a number of languages, including Japanese and simplified Chinese.

1.1.2 Client/Server

The client/server support as implemented in NetView for AIX Version 4 provides the ability to distribute the graphical user interface (GUI) and the processes that support it to other RISC System/6000 AIX systems.

This has the effect of reducing load and resource demand on the server system and on the network between the server and the GUI. Where possible, application programming interfaces (APIs) are unchanged by the client/server implementation, but there may be some impact on applications, because previously they only had to be concerned with running on a single system. Such applications can run exactly as before, but it may be better to re-design them to take full advantage of the client/server environment.

We describe the details of the client/server implementation and some of its implications in Chapter 2, "Client/Server Support in NetView for AIX" on page 7.

1.1.3 GUI Improvements

The graphical user interface (GUI) of NetView for AIX Version 4 retains the same general appearance as the previous releases. However, there have been several small enhancements made to improve usability, as follows:

Navigation Tree and View Stack Enhancement: The NetView for AIX network mapping capability differs from many other similar products in that it attempts to minimize the number of windows displayed at any one time. It does this by overlaying the existing subnet display when you double-click on a symbol to open the lower-level submap. This means that you may have many submaps open, but only have one or two of them displayed. The navigation tree and view stack are tools that allow you to easily move between open submaps. The navigation tree shows you all the open submaps and the parent-child relationships between them. The view stack shows you the parentage of the submap you are currently viewing.

Logically, each submap has one parent object. That is, when you double-click on a symbol that represents an object, the child submap of that object is opened. However, prior to Version 4, the parent of a submap was not its parent object, but the *symbol* representing that object. This had an unfortunate effect if an object had symbols on multiple submaps, since it meant that when you double-clicked on the symbol, the submap containing the parent of the new submap that you saw would not necessarily be the submap that you had just come from. In Version 4 the navigational relationships are preserved correctly.

Enhancements to Context Menus and Tool Palette: Three new functions allow an application to dynamically add menu entries to the object menu for a specific object, or to the tool palette. The function calls are OVwAddObjMenuItem, OVwAddObjMenuItemFunction and OVwAddToolPalItem. This new capability is used by the MIB Application Builder to dynamically add new monitoring applications to context menus.

There is also a new flag for tool palette registration entries that allows you to specify the order in which items appear.

Print Tool Enhancement: The print tool gives you the ability to print an image of a selected window. This tool can now be used with a color postscript printer, if you wish.

1.1.4 Manager Takeover Improvements

Manager takeover is a NetView for AIX function that allows you to subdivide management of the network among two or more systems. Manager takeover is not a full peer-to-peer communication system between managers, but rather a way of allowing one NetView for AIX to provide backup for another. Network container objects (such as IP subnetworks and segments) are associated with a primary and a fallback NetView for AIX. Both the primary and the fallback system have to discover the complete network, but the fallback system will have the objects in an *unmanaged* state. That is, it will not poll them. The fallback system also polls the primary system to check that it is available and, if it is not, NetView for AIX asks the user whether to manage the primary manager′s resources.

In NetView for AIX Version 4, several enhancements have been made to the way in which this process is implemented, as follows:

- Instead of monitoring to see if the whole of the system on which the primary manager is running is up, NetView for AIX now monitors an SNMP MIB table, which reports the status of the NetView for AIX daemons. There is a new SNMP subagent daemon, mgragentd, which provides this MIB table.
- There is a new submap that shows the NetView for AIX managers discovered in the network and shows their status.
- When the fallback NetView for AIX detects that the primary NetView for AIX has failed, it pops up a message asking whether the user wants to manage all the affected nodes. If the user clicks on **OK**, Version 3 would open submaps for all the affected nodes. With Version 4 the nodes are managed without opening submaps.
- When the primary NetView for AIX returns, another pop-up asks whether the resources should be unmanaged again. In Version 3 the user had to close all submaps to achieve this, but in Version 4 there is a **Close All** button to make this more convenient.

We do not cover manager takeover in more detail in this book, but you will find a full description of the Version 3 function in *Examples Using NetView for AIX*, GG24-4327.

1.1.5 Security Implementation

NetView for AIX Version 4 provides new security features that enable the network administrator to keep under control who is going to use the product and the capabilities of each user.

The facilities provided include the following:

- User authentication
- Controlled access to all NetView for AIX Version 4 items:

    Menu bar

    Context aenu

    Tool palette

    Command line commands

- Auditing
- Customization
- Integration of user-written programs

We describe the security feature in detail in Chapter 3, “Security” on page 25, including examples of how to configure it for your own use.

1.1.6 Event Rulesets

The processing of events in NetView for AIX is completely changed in Version 4, by the introduction of event rulesets. These provide the following capabilities:

- Correlation between different events, or the ability to treat two related events as one trigger
- Read and write access to additional information, external to the event stream itself (for example, the object database, MIB data and global variables)
- An override capability so that event severity and node status may be dynamically modified
- For all of these features, the ability to trigger on and have access to any of the information carried within the event

We describe the event stream enhancements in Chapter 5, "Introducing NetView for AIX Event Rulesets" on page 79. We also show a large number of ruleset examples in Chapter 6, "Examples of NetView for AIX Rulesets" on page 85 and Chapter 7, "Using the Collection Facility with Event Rulesets" on page 171.

1.1.7 Event Display Enhancements

There are several changes to the event display application, nvevents, in NetView for AIX Version 4, as follows:

- Integration of event rulesets. Correlation and filtering can now be done using ruleset processing. The filtering capability of Version 3 is still supported, but it has been functionally superseded by rulesets.
- Support for the client/server implementation. The event display application, nvevents, can now run on distributed client systems. This is achieved by means of a new daemon, nvserverd, with which each copy of nvevents establishes a socket connection. The nvserverd daemon is then responsible for distributing events as they arrive.
- Synchronization between different users. Prior to Version 4, any actions that a user made against events in an nvevents workspace were not seen by other users. Specifically this applied to adding notes or clearing events from the display. With Version 4 these changes can be broadcasted to all other NetView for AIX users. In addition, a user can make a severity or category change to an event, and it will also be reflected in all users′ displays.
- Color-coded card tabs. The card format event display now has a colored tab on the top of which indicates the severity of the event. The mapping of colors to severity is controlled by X-Windows resources. At the time of writing, the list format of the event display did not have color coding, but it was a planned enhancement.

1.1.8 Object Collection Facility

The object collection facility as implemented in NetView for AIX Version 4 provides important enhancements to the organization of network management. Collections can be created of nodes with similar characteristics, such as hardware, network address or object database information.

There are two tools provided with the object collection facility: the graphical user interface (GUI) and the collection APIs. The first supports all functions needed by the user to create, test, modify, delete any collection. The second one is helpful in implementing applications that use collections features.

We describe the collection facility in detail in Chapter 4, “The Collection Facility” on page 53, including several examples of how to create collections and sample code that uses the API.

1.1.9 Agent Policy Manager

The agent policy manager is a new feature of NetView for AIX Version 4 which simplifies the configuration and management of Systems Monitor agents. We describe the facility in more detail in Chapter 9, “Agent Policy Manager (APM)” on page 201.

1.1.10 Intention to Support SNMP Version 2

As part of the announcement of NetView for AIX Version 4, a statement of direction was made that SNMPv2 will be supported in the near future. The main reason for the delay of this support is the unfortunate confusion over the security aspects of the new SNMP Version 2 standards.

NetView for AIX Version 4 will use a dual stack approach, providing a WINSNMP API implementation to allow applications to have transparent access to SNMP Version 1 and Version 2 agents, while at the same time preserving the current OVSNMP API.

Chapter 2. Client/Server Support in NetView for AIX

From the very first version, NetView for AIX has been designed to permit multiple concurrent users. Prior to Version 4 this support was achieved by each additional user establishing an X-Windows session to the NetView for AIX machine. Internally, NetView for AIX consists of a number of background and foreground processes. When additional X-Windows users are added some of those processes are replicated. We can therefore divide the NetView for AIX processes into two categories, *user-specific* (that is, related to a single end user interface) and *common services* (that is, providing functions used by all users).

As the NetView for AIX configuration grows, it becomes desirable to distribute the functions across more than one machine. One approach to this was implemented in NetView for AIX Version 3 and Systems Monitor for AIX Version 2. Systems Monitor for AIX performs some of the network polling (for device status and performance thresholds) that would normally be the responsibility of NetView for AIX. There are two main benefits to this:

1. Reduced CPU and memory requirement on the NetView machine
2. Reduced network load, since polling occurs locally

With NetView for AIX Version 4 client/server support the distribution of function is taken one step further. Instead of the user-specific and common services processes all running on the same RS/6000, it is now possible to spread them across several RS/6000s. We expect this to deliver a number of benefits:

1. Reduced CPU and memory requirement on the main NetView for AIX processor, which in turn allows an increased maximum number of concurrent users.
2. Reduced network load, since the distributed EUIs do not require as much network bandwidth as the X-Windows sessions required previously.
3. More power to handle larger networks and more operators because having separate processors handling EUI and background tasks effectively gives us parallel processing.

These benefits also give us more flexible ways to configure NetView for AIX. For example, instead of installing one large RS/6000 to support (say) four users we could install two smaller machines and configure one as a server and the other as a client each with two attached users.

In this chapter we describe the internal structure of NetView for AIX, we look at some preliminary performance results and finally we consider the impact that client/server has on applications using the NetView for AIX EUI and API facilities.

2.1 NetView for AIX Internal Structure

First we will show the pre-client/server (Version 3) components of NetView for AIX and then we will describe how they are modified in Version 4.

2.1.1 Components of NetView for AIX V3

Figure 1 shows the main user-specific and common services parts of NetView for AIX Version 3.

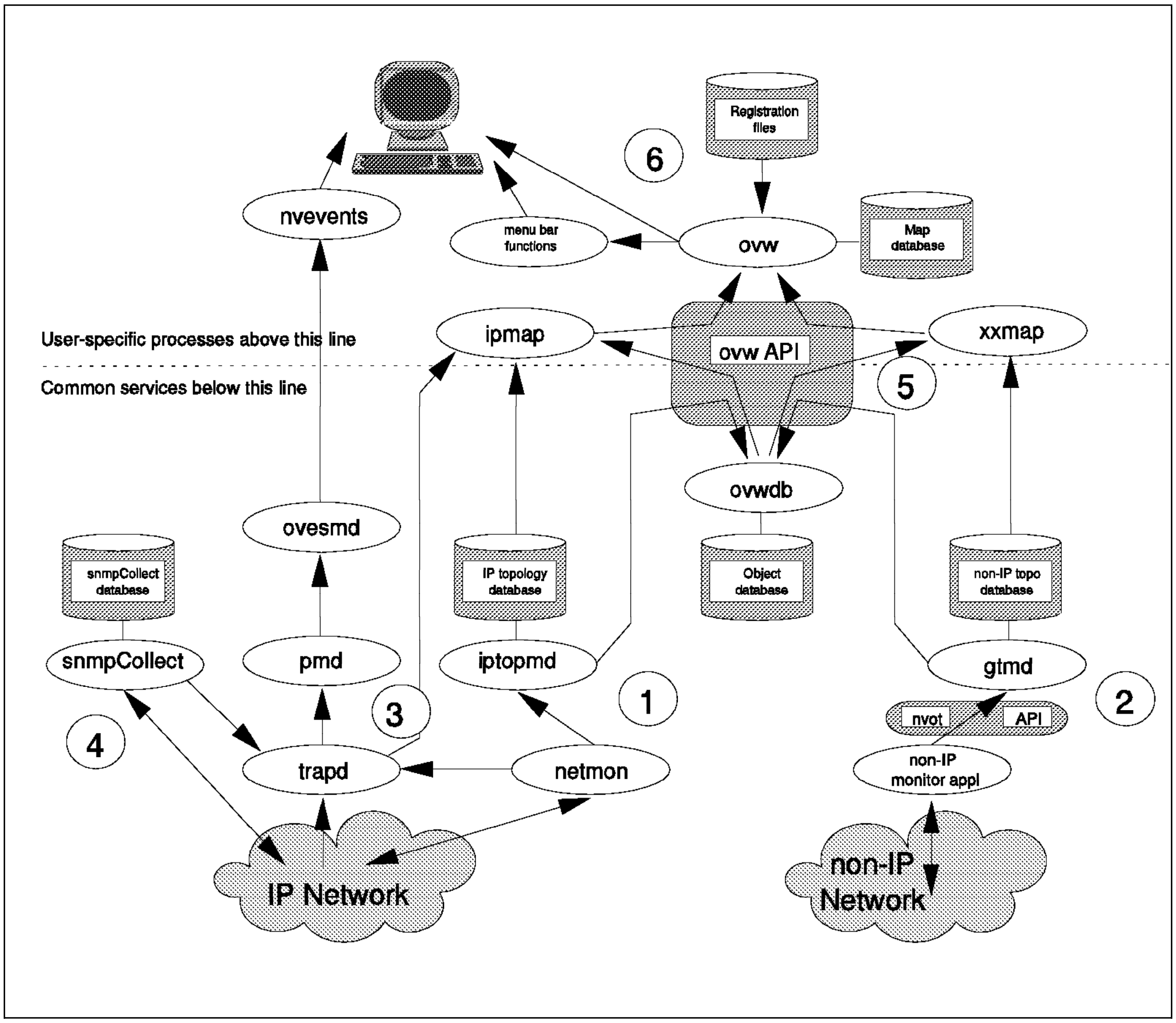


Figure 1. Components of NetView for AIX Pre-Client/Server. Note that some components and connections have been removed for clarity.

In the diagram, processes (both user processes and daemons) are shown as ellipses. There is only ever one instance of the common services processes (below the dotted line) but each new user ID creates another set of user-specific processes (above the dotted line).

We can divide up the components into the following series of process areas (numbered on the diagram):

1. IP network discovery and polling

   The netmon daemon is the engine for IP network polling. It discovers nodes in the network using SNMP requests and polls nodes for availability and configuration. As netmon detects changes in the network configuration it generates internal event records that are passed to the trapd daemon and it updates the IP topology database and object database. These two databases contain similar information but have different purposes. The object database contains one record for every network resource (node, interface, segment, etc.) whether they are IP resources or not. Associated with each record are a set of database fields describing the resource. All of this information can be queried and modified by API calls, so it can be used by any application within the NetView for AIX framework. By contrast the IP topology database is an unpublished format containing only information about IP network resources.

2. Non-IP network discovery and polling

   The non-IP components mirror the IP side. There is a closed-format topology database controlled by the gtmd daemon which is equivalent to the IP topology database controlled by iptopmd. The difference is that instead of the data being specific to one protocol it is in a generic format. An API (the NetView Open Topology (NVOT) API) is provided to allow applications to populate the non-IP topology database. Applications are required to provide their own version of the netmon daemon for network discovery and polling. Many other SystemView for AIX features use this facility, for example: LAN Network Manager, Lan Management Utilities and IHMP for AIX.

3. Event processing

   Events in NetView for AIX are mainly packaged as SNMP traps (the exception is events originating from the XMP API, which are OSI CMIP format). Traps are received by the trapd daemon, either from remote SNMP agents or from other NetView for AIX processes. These events are then merged with those from XMP by the pmd daemon and passed to the sieve agent, ovesmd. This implements a filtering process so that applications (for example, the event display application, nvevents) can receive specific event types.

4. MIB data collection and monitoring

   The snmpCollect daemon polls SNMP agents for for performance data. The data may be stored and/or compared with a threshold value.

5. The ovw API

   This API is at the heart of the NetView for AIX facilities for displaying network topology and configuration information. It is really divided into two parts:

   - *ovwdb*, which allows a program to query and modify information in the object database
   - *ovw*, which allows a program to query and modify the map database (that is, the contents of the topology views seen by the user)

   Notice that this second (map database) part is provided by the ovw program. This is a user-specific process, so calls to the API will be associated with a particular user′s display. To put this another way, if you want your code to update every user′s map you have to have a separate copy of the code running for each user. The corollary is that you cannot run a program to update or read map information if there is no EUI active.

6. The user interface processes

   When a user invokes the NetView for AIX EUI, the ovw process is started. The ovw process reads registration files to determine how to build the EUI menu bar and which other processes to launch. Three processes that are always invoked in this way are:

ipmap	Reads IP topology and object database information and uses the ovw API to create and maintain the IP network submaps (you will see the message: ″Synchronizing″ on the main NetView window when ipmap is updating the map database in this way)
xxmap	Performs the equivalent function to ipmap, but for non-IP resources
nvevents	Displays event records in a card or list format

There are also other user-specific processes that are non-persistent, such as the MIB Browser, MIB monitoring applications and other applications launched from the menu bar.

2.1.2 Components of NetView for AIX V4

It is possible to run NetView for AIX V4 in the same way as V3 - that is, with both the user-specific and common services processes on one machine. In fact, the internal structure of the product changes little when it is operating in client/server mode, as Figure 2 on page 11 shows.

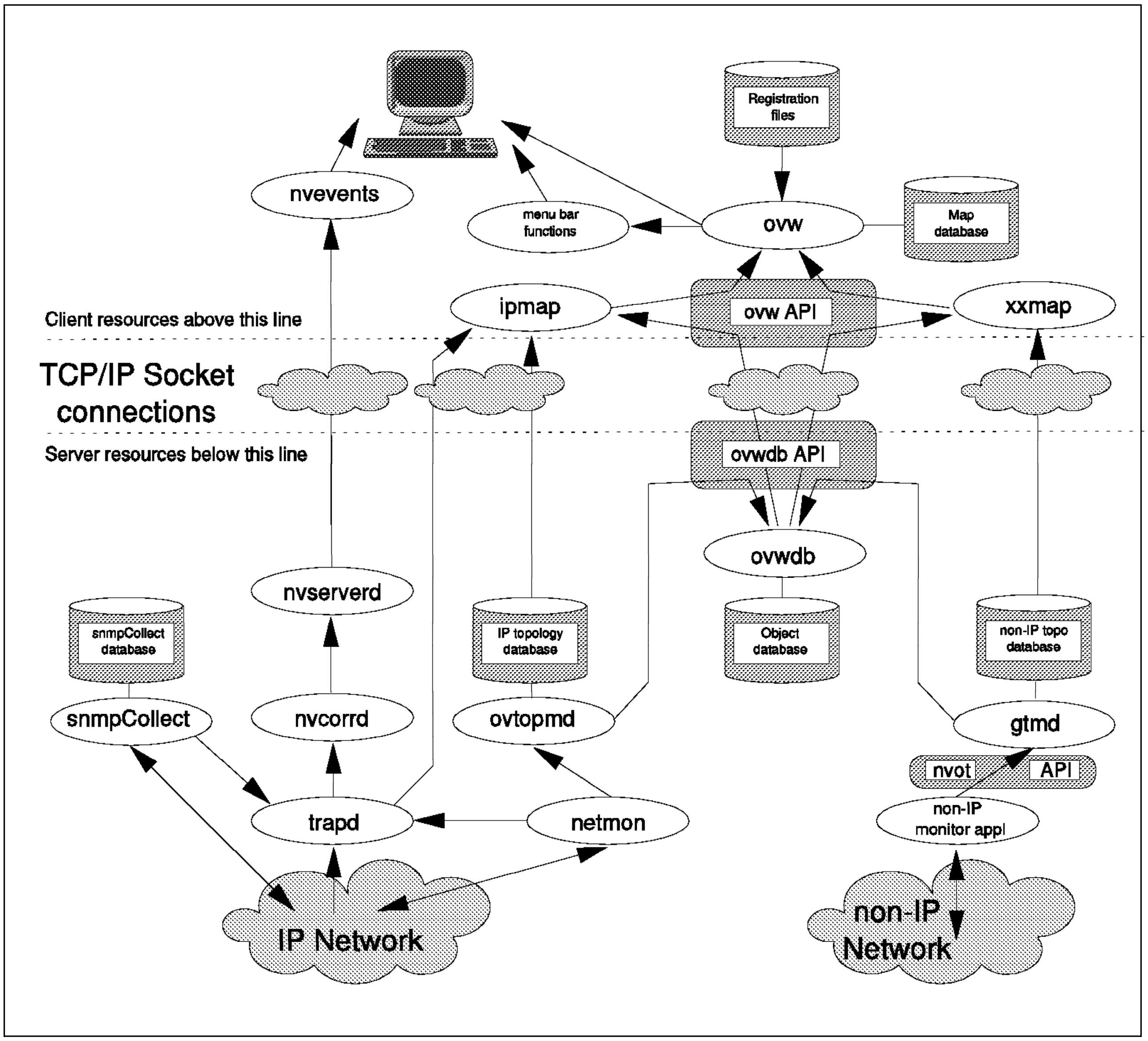


Figure 2. Components of NetView for AIX Client/Server. In the event-processing path the pmd and ovesmd daemons from Version 3 are still there, but we have omitted them from this diagram for clarity.

In general the only differences from the single system version are:

1. The user-specific processes are running on the client machine and the common services are on the server.
2. Process-to-process connections between client and server are encapsulated in TCP/IP sessions.

You should note that a client machine can support multiple concurrent users (using X-Windows to distribute the displays) and that client users can coexist with users directly connected to the server. It would, for example, be possible to have a system with twelve X-station users, six supported from the server and six from one or more clients.

There are several points to note from the internal process diagram (Figure 2):

- The registration files that control menu entries and EUI applications are located on the client machine. This means that any change to these must be installed on all attached clients. For example, if you install an application on the server that adds menu entries you will only see those menu entries on the client machine if you copy the registration files and the executables that they invoke.

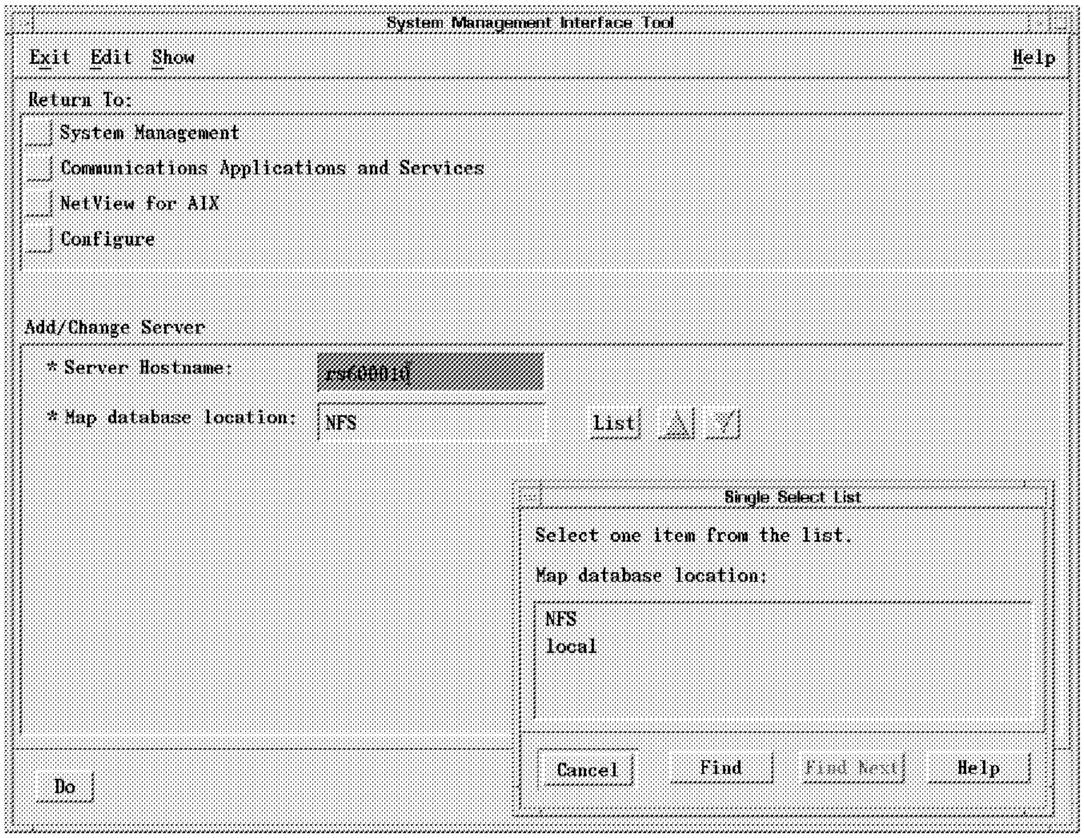
- The ovw API at the client machine is unchanged. Any application that uses this API in single system mode should be able to run unchanged on the client machine. On the server the ovw API is restricted in the same way as with Version 3; if no EUI is running only the object database can be accessed. Although, as we have said, many applications will run unchanged on the client, there are several reasons why this may not be straightforward:
  - If the application has common code that relies on all EUIs running on the same system it may need to be restructured or rewritten.
  - If the application employs a daemon using the ovwdb or nvot APIs, it will need to be restructured so that the daemon is installed on the server and the EUI function is installed on the clients.
  - There are software licensing considerations, since multiple copies of the code have to be installed.
- Although the map database is *logically* located on the client machine, there is an option to use NFS to mount it from the server. We discuss this option in more detail in 2.1.3, "Map Database Options."
- Menu bar applications that use SNMP (for example, the MIB Browser and graphing tool) will run on the client. They use SNMP community information from the /usr/OV/conf/ovsnmp.conf file and NetView for AIX will automatically mount this from the server via NFS so that it is consistent. However the agents that the applications are accessing may need community information updated to grant access to requests originating on the client machine(s).
- The daemons involved in event handling have changed. This is a result of the introduction of event rulesets (see Chapter 5, "Introducing NetView for AIX Event Rulesets" on page 79) rather than client/server.

2.1.3 Map Database Options

As we have seen (Figure 2 on page 11) the map database logically resides on the client machine. However this means that map customization changes are *only* applied to the client. Furthermore, that map will not be available for users at any other client, nor at the server machine.

NetView for AIX provides a way around these limitations. You can configure a client to use a map database mounted via NFS from the server. This allows maps to be centrally configured and used anywhere, at the expense of increased network load and reduced security. 2.2.3, "Testing Network Utilization with Client/Server" on page 17 discusses the performance implications.

You configure a client to use local or NFS mounted maps by selecting **Configure→Add/Change Server** from the clients in NetView for AIX SMIT screen. Figure 3 on page 13 shows the SMIT configuration screen.



Return To:
System Management
Communications Applications and Services
NetView for AIX
Configure

Add/Change Server
* Server Hostname: rs600010
* Map database location: NFS

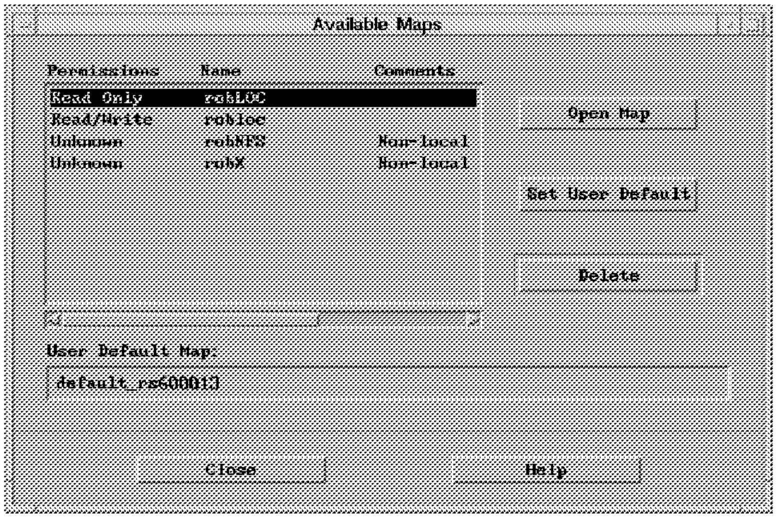
Select one item from the list.
Map database location:
NFS
local

Figure 3. Defining Client Configuration Options

Although the map database may now effectively be divided (some maps may reside on client machines and others on the server) it is still treated as if it were a single entity. To manage this, NetView for AIX enforces the following rules:

1. Each map name can only exist once. Even though it cannot access the contents of the maps, the server is aware of what maps exist on each of the client machines. It prevents the creation of duplicate map names.
2. Only one user can have a map open in read/write mode. This restriction also existed in previous versions of NetView for AIX. It is now enforced across clients (for NFS-mounted maps) and within clients (for multiple users on a single client).

Figure 4 on page 14 shows the screen displayed when you select **File→Open Map** from the NetView for AIX menu bar.



Permissions	Name	Comments
Read Only	robLOC	
Read/Write	robloc	
Unknown	robNFS	Non-local
Unknown	robX	Non-local

User Default Map:

default_rs600013

Figure 4. Map Selection List

In this case we are running on a client machine with local maps. Two maps exist on the client, but we only have update access to one of them. The other two maps listed are either on the server or on another client, and therefore not available to us. Notice that the default map name has changed for client maps. Instead of simply being called default it is now default_xxxxxx where xxxxxx is the machine where it exists. For EUIs running on the server the default map name remains default.

2.2 Performance Considerations

We have already mentioned the performance advantages that we anticipate from the client/server feature.

1. We expect to be able to support more concurrent users, since the user-specific code (to be precise, the memory and CPU cycles used by that code) is spread across more systems.
2. We expect to need less network bandwidth to support multiple users, since the client/server connections should transfer less data than the X-Windows protocol and they should also be less seriously affected by poor response.

Before we look in more detail at the effect of client/server on resource utilization, we will briefly explore the murky world of performance measurement and prediction.

2.2.1 How Much is Enough?

The sort of questions we *want* to be able to answer are:

- What size machines do I need to run NetView for AIX for a network of ″X″ nodes and ″Y″ concurrent operators?
- What will be the network load from monitoring ″Z″ nodes?

The problem with these questions is that they expect *absolute* answers. In reality there are many unknown variables, which make it impossible to give such an answer. Some examples of the unknown variables are as follows:

- As CPU load increases, responsiveness will decline. But how do you define the point at which it becomes unacceptable?
- How do you assess CPU utilization for an application that is driven erratically by a user using a graphical user interface. To put it another way: what is a ″typical″ user load?
- Memory utilization is a critical factor in NetView for AIX, because it maintains a lot of network resource data in memory. You can measure memory utilization accurately, but AIX provides a virtual memory, paging system. How do you predict the point at which page faults degrade performance to an unacceptable level?
- Network problems may affect the polling processes of NetView and can seriously affect performance of a normally healthy system. How do you judge the extra capacity needed to cope with these crises?

The normal response to these difficulties is to create *rules of thumb* based partly on empirical data and partly on experience. At the time of writing, these rules of thumb have not yet been derived for NetView for AIX V4. However, we have seen that the internal design has not changed greatly from Version 3, so it seems reasonable to extrapolate to a limited extent from that version. The currently accepted rules of thumb for sizing NetView for AIX Version 3 are described in *SystemView for AIX: Sizing Considerations*, GG24-2586.

In this project we did not have the time or resources to try to create rules of thumb for sizing NetView for AIX V4, but we did perform some simple tests to verify that we were realizing the expected performance benefits from client/server.

2.2.2 Testing Memory Utilization with Client/Server

The memory requirements for NetView for AIX V3 are tied to two factors:

- The number of managed resources in the network
- The number of concurrent users

We expect that the latter factor will be greatly decreased in Version 4 by placing the users on client machines. We performed a simple test to verify this behavior.

We used Performance Toolbox/6000 (a SystemView Feature) to measure memory and page space utilization for the system as a whole and for individual NetView for AIX processes. We took the measurements with 0, 1, 2 and 3 users active, firstly via X-Windows sessions and secondly from a client machine with local maps. The results we obtained are shown in Figure 5 on page 16.

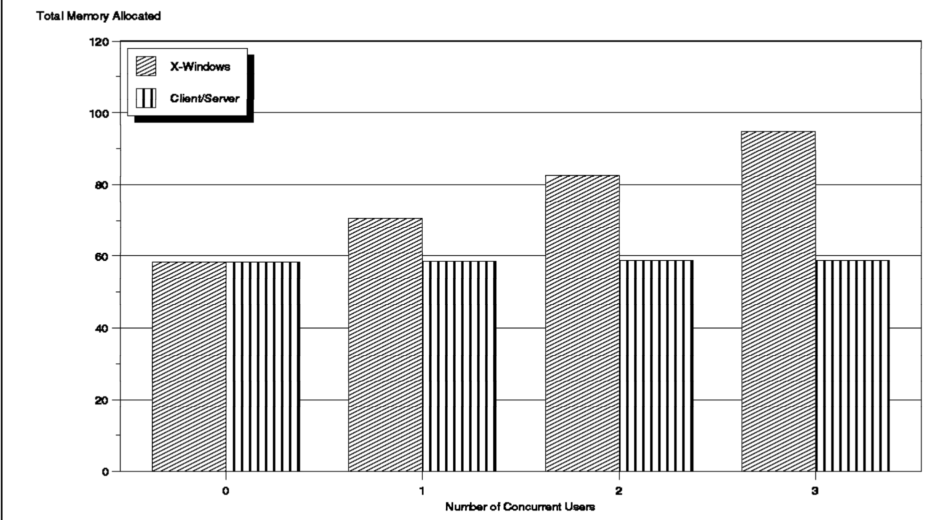


Figure 5. Comparing Memory Utilization in Client/Server Mode

We see the expected results: when using X-Windows the user-specific processes add a memory delta of about 12Mb per additional user. When the users are accessing the system via client/server we see a negligible increase (in fact, we measured an increase of about 128Kb for each additional user, shared among five of the NetView daemons).

We also measured memory utilization on the client system and saw an increase in memory allocation with each additional active user. The delta was, as expected, the same as we saw on the server in the X-Windows case (about 12Mb).

What Does This Prove?

This test merely shows that using client/server has the expected effect of sharing the memory requirement between server and client. It appears that the memory used by NetView is similar in each case (although the *total* memory requirement will be larger because it must also include the base memory needs of AIX on each client machine).

The test does *not* give any idea of how much real memory is needed (for that we would need to know how volatile the memory is, and hence what proportion of it can be paged-out without degrading performance). The test also takes no account of network size or user activity. Network size is the major factor in deciding memory requirements. You should refer to *SystemView for AIX: Sizing Considerations*, GG24-2486, to see how network size affects memory demand. At the time of writing the figures in that book are based on NetView for AIX Version 3, but they make a good starting point for sizing estimates.

Although we see from this test that the cost of additional clients in terms of memory is small, there will also be a cost in CPU cycles. We did not attempt to assess this, indeed it is difficult to do so, since it is dependent on many different factors. As more experience is gained with different client/server configurations, we will be able to create rules of thumb for this.

2.2.3 Testing Network Utilization with Client/Server

For this test we wanted to get a feel for the comparative network cost of the following three options for connecting remote users:

1. Single system with X-Windows access
2. Client/Server with local maps
3. Client/Server with NFS-mounted maps

We used DatagLANce to perform the data collection. DatagLANce is a OS/2-based LAN analyzer with powerful facilities for data collection, interpretation and display. In our case we only used the frame capture facilities to trace all the activity to and from our client and server machines. We then employed the IBM LAN Doctor service to get reports showing a breakdown of network activity. LAN Doctor offers a detailed LAN problem determination and analysis service, including application profile modelling and load prediction.

We needed to make sure that we captured a similar set of end-user activities for each of the three cases. We therefore designed a standard sequence of plausible user actions and repeated them for each of the cases. Table 1 shows the script we used.

Table 1 (Page 1 of 2). Test Script for Network Data Collection

Time into test	**Operation**
	Start the NetView for AIX EUI and wait for initialization to complete (map synchronization complete and nvevents display active). We did not capture network activity during this period.
0 seconds	Open sequence of submaps: IP Internet, Network, Segment, Node.

Table 1 (Page 2 of 2). Test Script for Network Data Collection

Time into test	Operation
30 seconds	Return to Root submap using the view stack, open MLM Managers submap and one of the MLM domain submaps. Return to the Segment submap previously opened using the Navigation Tree.
60 seconds	From another RS/6000, send three traps that appear as event cards in the nvevents window.
90 seconds	Using the Search option, create a static events workspace containing all events from a single node (5 event cards in each case)
120 seconds	Select a node from the segment submap and start a monitor graph of interface traffic.
150 seconds	Using the same node, start the MIB Browser, select Mgmt→MIB2 →System and retrieve all the system information from the SNMP agent on the selected node.

The traffic recorded between the client and server machine for this script is summarized in Figure 6.

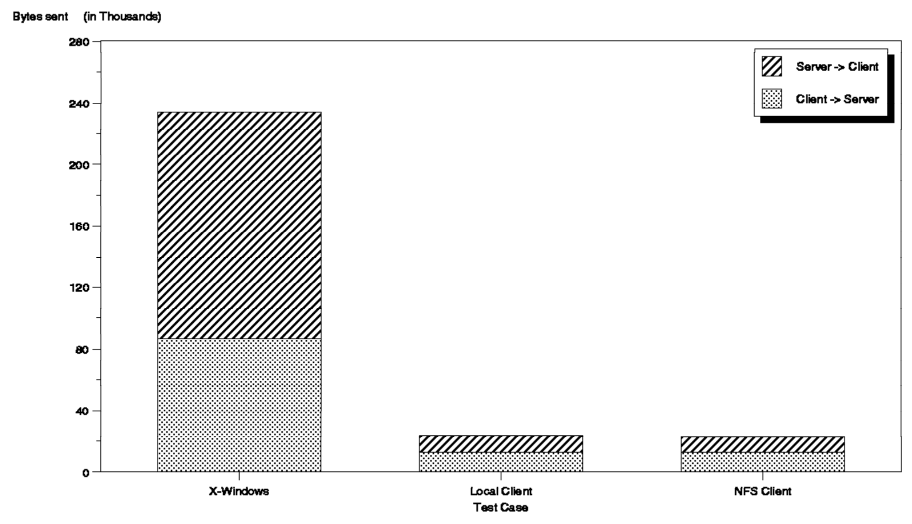


Figure 6. Network Utilization for "Typical" User Script. Note that the "client" is the user's machine and the "server" is the NetView machine, even though this should technically be reversed for the X-Windows case.

The comparison between the X-Windows case and the client with local maps is as expected. There is about a ten-fold difference between the amount of data transferred. In fact, the raw figures do not fully illustrate the differences. In the X-Windows case any delay in the network is perceived by the user, whereas the client user may not be aware of it. For example, a user may be unhappy with a half-second delay when opening a menu or displaying a window, but will not notice a similar delay in (say) a change in the status color of a node.

One aspect of Figure 6 that we initially found puzzling, is that the case with local maps and NFS-mounted maps are almost identical. We had expected to see more activity due to NFS access when the submaps were opened. The reason for this discrepancy is that all the map actions taken by the user were *reads*; there were no status changes, so ipmap did not have to update the map database. NFS provides a caching function, so data that is unchanged may be read locally. We had just started NetView for AIX at which point the synchronization process causes ipmap to update status on all IP map symbols. The cache would therefore be up-to-date, hence the lack of extra network activity. To test this theory we did another data capture, this time just capturing the EUI startup (from the initial command until map synchronization completed and the nvevents display was active). We used the worst case situation of a new map, so that the startup included the complete map build in each case. The result of this test is shown in Figure 7.

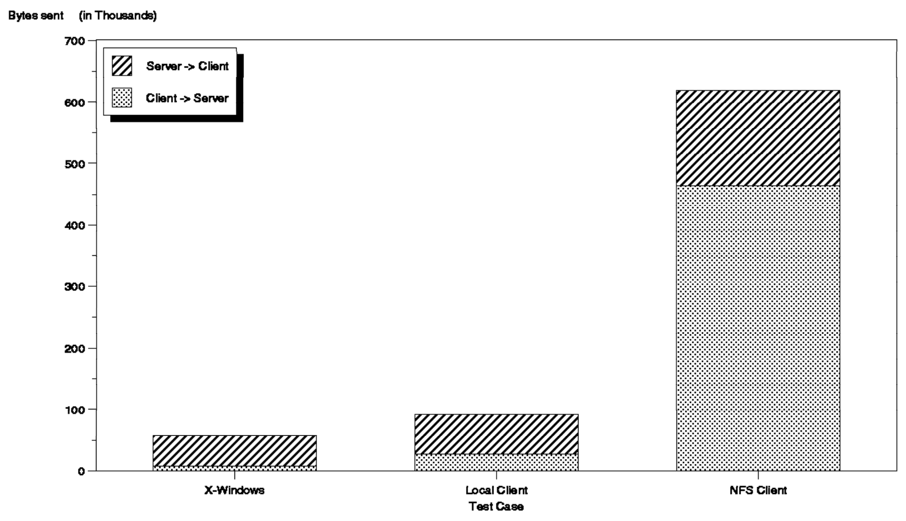


Figure 7. Network Utilization for EUI Startup

As expected, we see a lot of activity in the NFS case, as ipmap on the client updates the map database on the server. This was also reflected in the time taken to start up. The NFS client took almost three minutes, compared with a little over one minute for the other two cases. One other aspect to this is that using NFS may give erratic performance, depending on whether the map data in the NFS cache has been flushed by newer data from other NFS activity.

What Does This Prove?

These tests show that, as expected, the local map option has the lowest network cost. In a LAN environment the extra traffic of X-Windows is probably not significant, but across a WAN link it would be. The test also show that the increased convenience of using NFS for the maps is offset by slow initialization and potentially erratic performance.

The tests do *not* give any general rules of thumb for the network cost of the three configurations. They are based on a synthetic user load which may bear little resemblance to reality.

2.3 API Considerations with Client/Server Concept

In *Examples of Using NetView for AIX*, GG24-4327, the example program wteuiap6 was discussed. This program uses the ovw and ovwdb APIs described in 2.1.1, “Components of NetView for AIX V3” on page 8 and 2.1.2, “Components of NetView for AIX V4” on page 10, to provide an easy way to create and modify network topology displays in NetView for AIX.

In this section we will discuss what needed to be done to modify the way that wteuiap6 is implemented in order to make it operate in a NetView for AIX Version 4 client/server configuration.

2.3.1 The wteuiap6 Sample

Figure 8 shows the different components of wteuiap6.

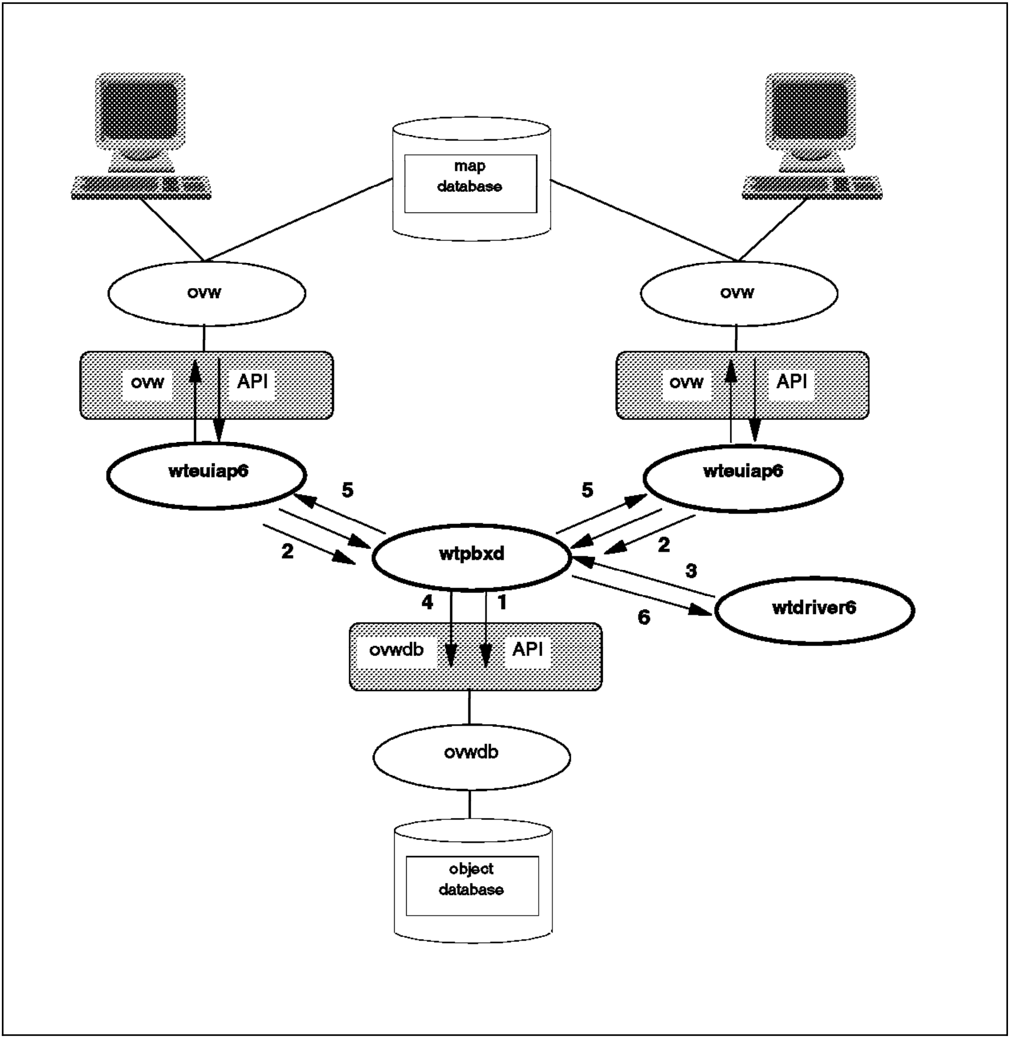


Figure 8. Components of wteuiap6

As Figure 8 shows, there are three programs within the application, as follows:

wtdriver6 This is the command line program that the user enters to invoke the wteuiap6 functions. It takes a series of arguments that define the action to take, for example: to add nodes or submaps, make connections, set status, etc.

wtpbxd	This is a daemon that uses the ovwdb API to update the NetView for AIX object database. The wtpbxd daemon is defined to NetView for AIX by means of a *local registration file* (lrf). It is started by ovspmd at the same time as the other NetView for AIX daemons.
wteuiap6	This is a background process that uses the ovw API to update the map database (and thus the users′ display). Note that as the ovw API is supported by the ovw process, there is one copy of wteuiap6 for each copy of ovw that is running. The wteuiap6 process is defined to NetView for AIX by means of an *application registration file*.

The sequence of events that takes place when wteuiap6 is used is as follows (the numbers refer to the numbered arrows in Figure 8 on page 20):

1. When wtpbxd starts it opens the ovwdb API and waits for NetView for AIX end user interfaces to start.
2. As each end user interface is started, a new copy of wteuiap6 is also started. These send a message to wtpbxd to signal their existence. The wtpbxd daemon keeps a table in storage with details of all the current EUIs, including details such as the map name and permission (Read/Write or Read Only).
3. Later, a wtdriver6 command is entered by a user or from within a shell script. The arguments are parsed and then sent in a message to wtpbxd.
4. The wtpbxd daemon can perform directly any wtdriver6 requests that only need access to the object database, such as queries or updates of database fields.
5. If the wtdriver6 request needs to update the map database, wtpbxd sends it to each copy of wteuiap6, which executes the request by calling the ovw API (the default behavior is to update every EUI, but the user can restrict it to a specific one by using a wtdriver6 option).
6. Finally the user receives a message indicating the success or failure of the request.

2.3.2 Updating wteuiap6 for Client/Server

The ovw and ovwdb API calls that the wteuiap6 components use are unchanged by the client/server environment. They are all available on both the client and server machines. However this does not mean that the application does not need changes. If we project the wteuiap6 structure onto the NetView for AIX client/server environment we can see that:

- The wtpbxd daemon is a central coordinating function that logically should be placed on the server.
- A wteuiap6 process is associated with each EUI. It will therefore be running on the system where the EUI itself is running, which may be client or server.
- The wtdriver6 command may be invoked on either a client or server system.

We made the following changes to the wteuiap6 components to support the client/server structure:

- When a new copy of wteuiap6 registers itself with wtpbxd it now has to send the message to the server system. We used the new OVDefaultServerName() API call to find out the IP address of the server.

- We added the client system IP address to the table of active copies of wteuiap6 that wtpbxd maintains.
- We added a parameter to wtdriver6 to identify the system where wtpbxd is running (that is, the server system).

In addition to the wtdriver6 command interface we provided a menu driven driver. Upon startup this attempts to connect to wtpbxd on the local host and if this fails will request the host name on which wtpbxd is running. This can be called either from the ovw menu bar by selecting WTEUIAP6→wtmenu6, or from the command line by entering `/usr/OV/raleigh/wtmenu6`. It is not a requirement that ovw should be running on the same system as wtmenu6.

2.3.3 Installation

You can get the wteuiap6 sample via anonymous FTP. Refer to Appendix A, "How to Obtain the Samples in this Book" on page 257 for details.

Installation of the modified package is performed by running the command `make install`. This will install the executables into /usr/OV/raleigh and place the registration files in the relevant directories. It will also give the option to install the necessary files on clients that are recognized by the server.

There are also some shell script samples that demonstrate the abilities of wteuiap6. To try them, run either buildIDNX or buildMQSeries on the NetView for AIX Version 4 server system.

2.3.4 Configurations

These applications have been tested in client/server operation with the following permutations with both NFS and local client configurations.

- All Server Configuration

  When operating in an all server configuration there is no difference between wteuiap6 on NetView for AIX Version 4 and NetView for AIX Version 3.

- Client driver - Server applications.

  With wtdriver6 on the client and the wteuiap6, wtpbxd and ovw applications running on the server.

- Client applications - Server daemons.

  With wtdriver6, wteuiap6 and ovw on the client and wtpbxd on the server.

- Client application - Server daemons and driver.

  With wteuiap6 and ovw on the client and wtdriver6 and wtpbxd on the server.

2.3.4.1 Example of Stat Option

The status of wteuaip6 connections to wtpbxd can be seen from the wtmenu6 panel and will appear as shown in Figure 9 on page 23. In this example rs600010 is the server with IP Address 9.24.104.109. There are two clients active, 9.24.104.28 (which uses NFS-mounted maps) and 9.24.104.26 (which uses local maps). Each of the clients is serving two X-Stations. One of the X-Stations, itsoxst43, is using the server1 map in read-only mode. This map is also open in read-write mode on the HFT display on the server. Note that this configuration is only possible for a client with NFS-mounted maps. The user on itsoxst49 (connected to the client with local maps) would be able to see that the server1 map exists by selecting File→Open Map, but he would not be able to open it, because it is on the server.

```
SHOWING STATUS OF DAEMON
pbxd: Statistics:
      Total Calls: 22
            With a bad magic number: 0
            With a bad version number: 0
            With a bad command number: 0
            Which leaves 22 good calls
EUI Table, 5 entries
Entry Mode  Map              X-Display        Application       Pid    IP Address.Port
[  0] RW    nfs1             itsoxst47:0.0    EUI Application 6  44386 9.24.104.28.3074
[  1] RW    local1           itsoxst49:0.0    EUI Application 6  43213 9.24.104.26.110
[  2] RW    local2           itsoxst46:0.0    EUI Application 6  46807 9.24.104.26.1173
[  3] RW    server1          rs600010:0.0     EUI Application 6  46055 9.24.104.109.2637
[  4] RO    server1          itsoxst43:0.0    EUI Application 6  51627 9.24.104.28.4256
Press any key to continue
```

Figure 9. Wtmenu6 Status

There are four open maps, one of which (server1) has a read-write and read-only connection. All of these maps can be modified using wteuiap6 or wtmenu6 from the active EUIs or any other machine connecting to wtpbxd on rs600010.

Chapter 3. Security

This chapter discusses the security features built into NetView for AIX Version 4. The examples will show you how to add new users and how to group them accordingly to different NetView for AIX Version 4 maps and applications access policies.

3.1 Why Network Management Security?

There are basically five reasons why the new concept of network management security has been introduced in NetView for AIX Version 4:

Authentication An AIX system manager has different needs from a network manager, so it is a good practice to keep their environments separate. To obtain this separation you must use different authentication steps for AIX and for NetView for AIX Version 4.

Access Control Different NetView for AIX Version 4 operators may need access to different NetView functions. You can modify policies concerning:

- Menu bar Items
- Tool Palette
- Command line commands
- Context menus

Audit Security is nothing without logging. The network administrator must have the knowledge of what happened on the network regarding access violations, normal flow of operations, login/logout procedures, administrative changes and so on.

Administration All of this must be handled in an easy way, which means a simple end user interface, with contextual help, etc.

Customization Besides the security aspect, removing items from menus and limiting command line options offers also the considerable benefit of creating customized environments. For example, it can simplify the work of non-skilled operators by reducing the available menu options to just those that the operator uses.

3.2 Security in NetView for AIX Version 4

Previous versions of the product used standard AIX security features (user ID validation, file permissions, etc.).

In Version 4 a new concept of application security was introduced.

With this new concept, three possible levels of security are available:

Minimal Uses only standard AIX features, same as the previous version (default at startup).

2 Party NetView for AIX Version 4 takes care of security, authentication and validation.

3 Party	A third, dedicated, application validates users. A sample application may be IBM Network Security Program SLC or any other application supporting GSS API.

3.3 Terminology

These are the main terms we will use:

User	*Not* (necessarily) an AIX user. A NetView for AIX Version 4 user may not have its correspondent in AIX (that is, with an entry in /etc/passwd). You can log in to NetView for AIX Version 4 with a user ID that is not also an AIX user ID , but you still must have an AIX user ID just to reach an AIX shell. Putting this the other way around, an AIX user may not be able to login to NetView for AIX Version 4 unless explicitly authorized (root authority is not enough).
Security Registration Files (SRF)	These files contain the list of elements to be secured. They are provided by the application developers based on which elements they want to be secured.
Domain Group Profiles	These files contain definitions for ″can do″ and ″can′t do″ for every single application. Domain Group Profiles for all standard NetView for AIX Version 4 functions are included in the product.
Group	*Not* (necessarily) an AIX group. A NetView for AIX Version 4 group is a correlation between NetView for AIX Version 4 user IDs and SRFs. A user may belong to multiple groups (and choose one during login) and an SRF can be associated with multiple groups.
API	The security feature provides a simple application programming interface (API). Any application can use new the API calls to implement security functions.

3.4 How to Create a Trusted Environment

In this section we show how to set up a trusted environment using the 2 party level of security of NetView for AIX Version 4. The detail of configuring profiles and user IDs is only part of the process. It is equally important to plan the configuration in advance, to ensure that it meets your requirements.

We recommend that you follow the following sequence of steps:

1. Define (on paper) which level of security you need to implement.

   Ask yourself the following questions:

- *Do I really need Network Management Security?* You only need to impose security if you want to protect NetView for AIX from its users. If all the users of NetView for AIX are trusted, administrative, users you do not need additional security above the AIX login. In this case, you can use the default level of NetView for AIX Version 4 security (minimal).
- *Have I already installed a security application with GSS API?* If the answer is yes, use the GSS API application as a 3 party security server. Inform the GSS security application manager of your needs.
- *Am I going to use NetView for AIX Version 4 internal security?* If the answer is yes, decide:

  Who is going to manage user IDs for this additional environment?

  Who is going to implement the policies?

  How will this be documented?

In our example we will assume this last situation and we will implement NetView for AIX Version 4 internal security.

2. Define (on paper) which level of auditing you need to implement

   Are you interested in auditing configuration changes?

   Are you interested in auditing the accesses to functions?

   Are you interested in auditing login/logout activities?

   In general it is good security practice to log too much, rather than too little, even if it means using up a lot of disk space. Disk space is cheap these days. Of course, logging too much can also result in a harder search for the right information when needed.

   You can start initially with full auditing and keep an eye on the growing rate of your files. If it is unacceptable, you can later restrict the logging options.

3. Define (on paper) your groups, users and rules.

   The product comes with two predefined users, each of which is the sole member of a group:

   - Group:*Oper* user: *operator* for basic network operation functions
   - Group:*SrAdmin* user: *admin* for network administrators

   We recommend using these two groups as a starting sample and then adding your own. Remember that a user can be assigned to more than one group, which allows him to access NetView for AIX Version 4 with different privileges.

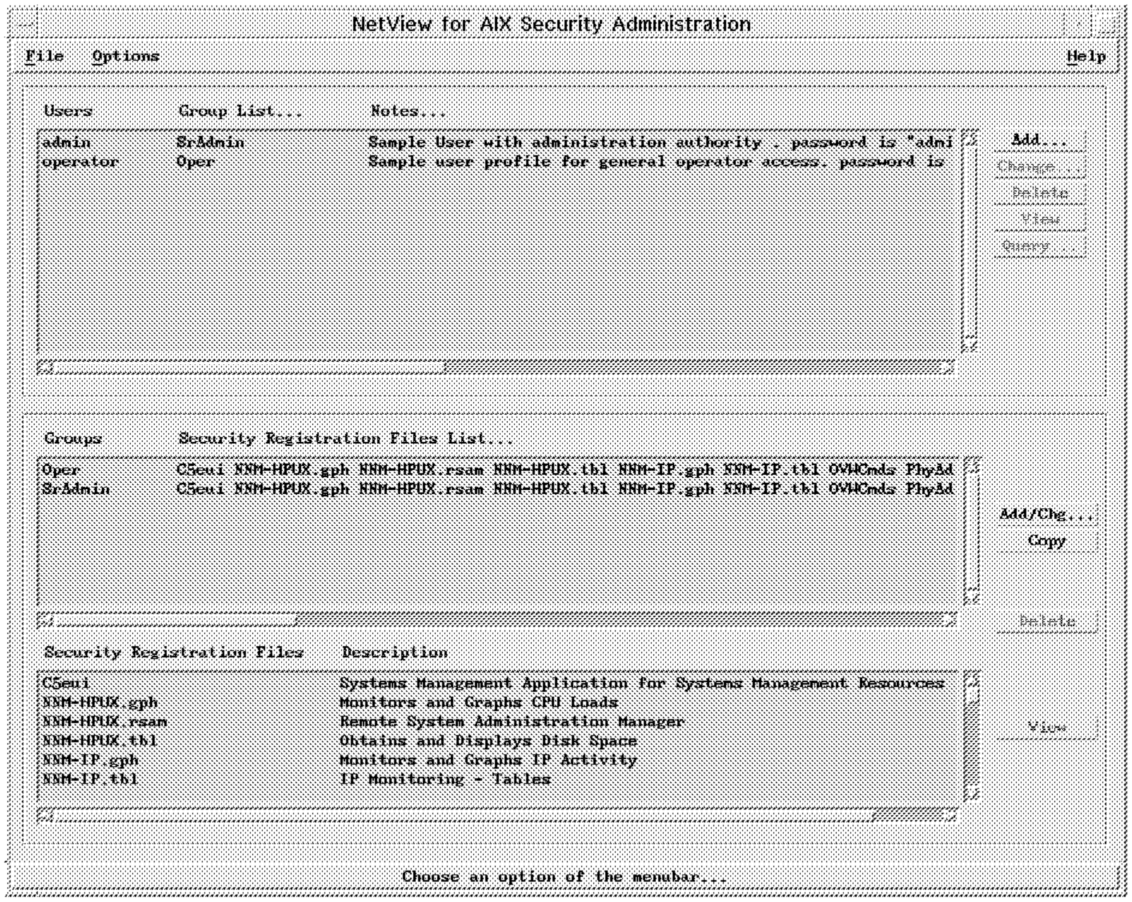
   Change Default Passwords !

   The default passwords (admin and operator) are trivial and can be found in the NetView for AIX manuals. Be sure to change default passwords as soon as you go in production.

4. Start customizing NetView for AIX Version 4

   Log in to AIX as root and start the X-Window interface (it is also possible to delegate security authorization to a different user, see *NetView for AIX Version 4 Administrator's Guide* SC31-8168).

   From a shell prompt execute the command `/usr/OV/bin/nvsec_admin`. The panel in Figure 10 on page 28 will be shown.



Users	Group List...	Notes...
admin	SrAdmin	Sample User with administration authority . password is "admi
operator	Oper	Sample user profile for general operator access. password is

Groups	Security Registration Files List...
Oper	C5eui NNM-HPUX.gph NNM-HPUX.rsam NNM-HPUX.tbl NNM-IP.gph NNM-IP.tbl OVWCmds PhyAd
SrAdmin	C5eui NNM-HPUX.gph NNM-HPUX.rsam NNM-HPUX.tbl NNM-IP.gph NNM-IP.tbl OVWCmds PhyAd

Security Registration Files	Description
C5eui	Systems Management Application for Systems Management Resources
NNM-HPUX.gph	Monitors and Graphs CPU Loads
NNM-HPUX.rsam	Remote System Administration Manager
NNM-HPUX.tbl	Obtains and Displays Disk Space
NNM-IP.gph	Monitors and Graphs IP Activity
NNM-IP.tbl	IP Monitoring - Tables

Choose an option of the menubar...

Figure 10. NetView Security Administration Main Panel

Let us start to familiarize ourselves with this panel.

The first box (Users) lists the users and their associated groups with comments. If the system has just been installed only the two default users are here: operator and admin.

The second box (Groups) lists the groups defined and the associated SRFs. We′ll see later where this correlation is set. Again, if the system has just been installed only the two default groups are here: Oper and SrAdmin.

In the last box (Security Registration Files) there is the list of the pre-defined SRFs. You can scroll through the list and see that all NetView for AIX Version 4 basic functions are there. In this example we won′t create a new SRF, we will use the standard ones.

5. Create any new groups we want to define.

   In our example we′ll add two new groups, each with one user, and a third user able to log in in both groups.

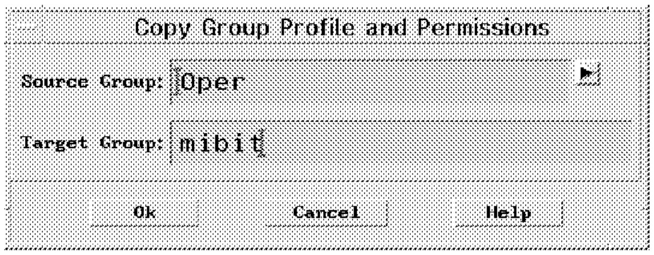
   Let′s think about a very low level user only able to see the topology and the status of the network but without any interaction available; we will give him a user ID of lluser.

   The second user will be someone that will be responsible for handling MIBs, so he will be able to load MIBs, browse them, make MIB applications and so on. We will give him user ID mibuser.

   These two users will be put in two new groups: llfunct and mibit.

A third user, with ID llormib will have the option to choose his role during login.

To create our two new groups we will use the Oper group as a template and take off all the functions we don′t need. We can start by copying the Oper group to our new groups by selecting Oper in the Groups box and then selecting **Copy**. The panel shown in Figure 11 will appear. Enter mibit in the Target Group field and click on **Ok** to add the new group.


Copy Group Profile and Permissions

Source Group: Oper

Target Group: mibit

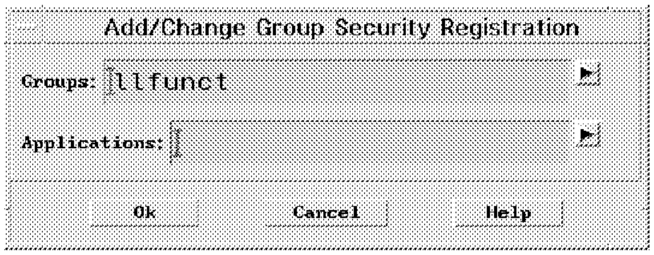
Figure 11. Copy Group Panel

We repeated this step for the other group, llfunct.

6. Configure application access for the user groups.

Now we have created the new groups, but the access they provide is identical to the group that we copied, Oper. Next we need to grant authorization to users in the groups by selecting the group from the list in the Groups box and selecting **Add/Chg**.

The panel shown in Figure 12 will appear.

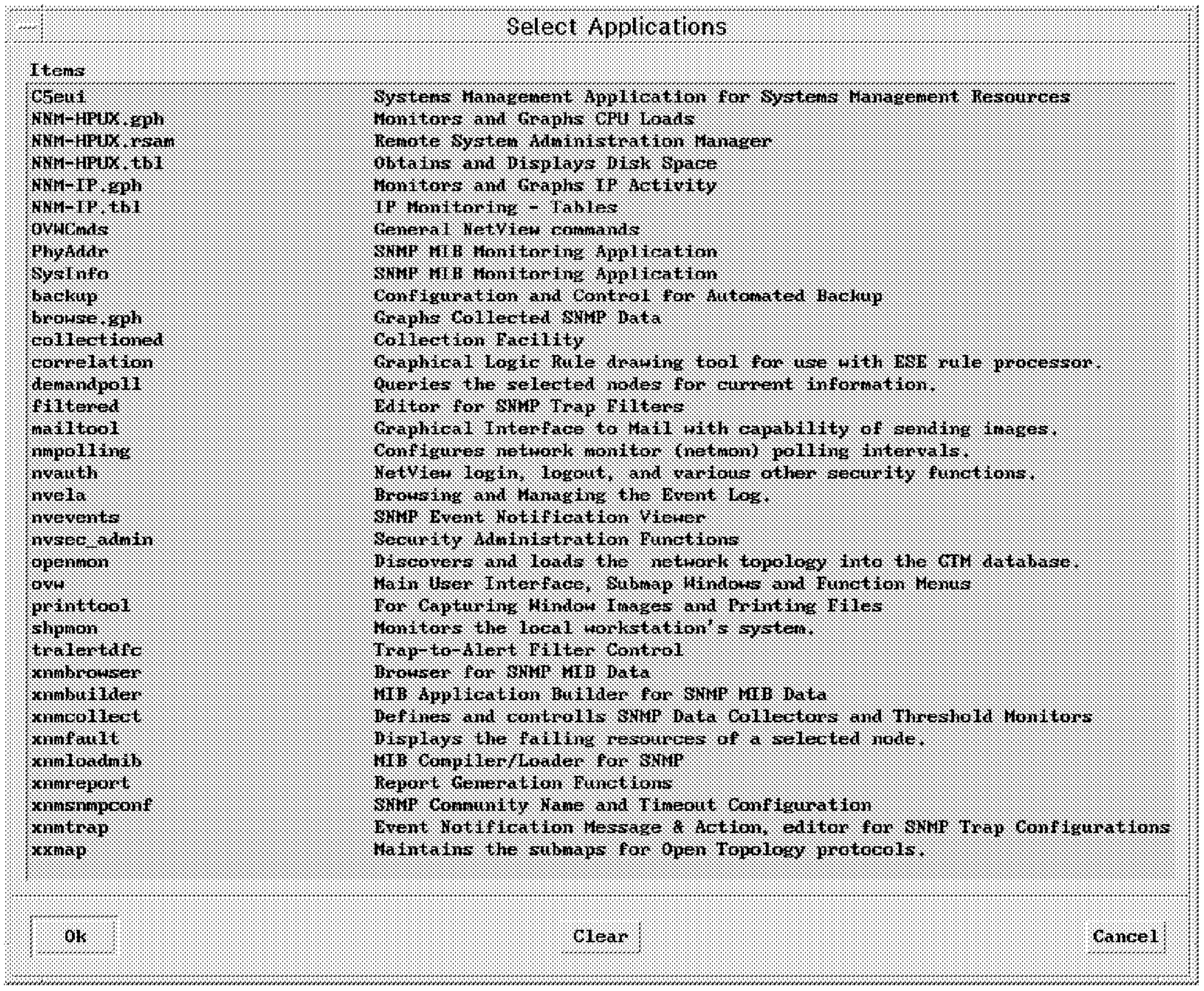

Add/Change Group Security Registration

Groups: llfunct

Applications:

Figure 12. Add/Change Group Security Registration Panel

Selecting the right arrow near the Applications field, we can see the list of applications available, as shown in Figure 13 on page 30.

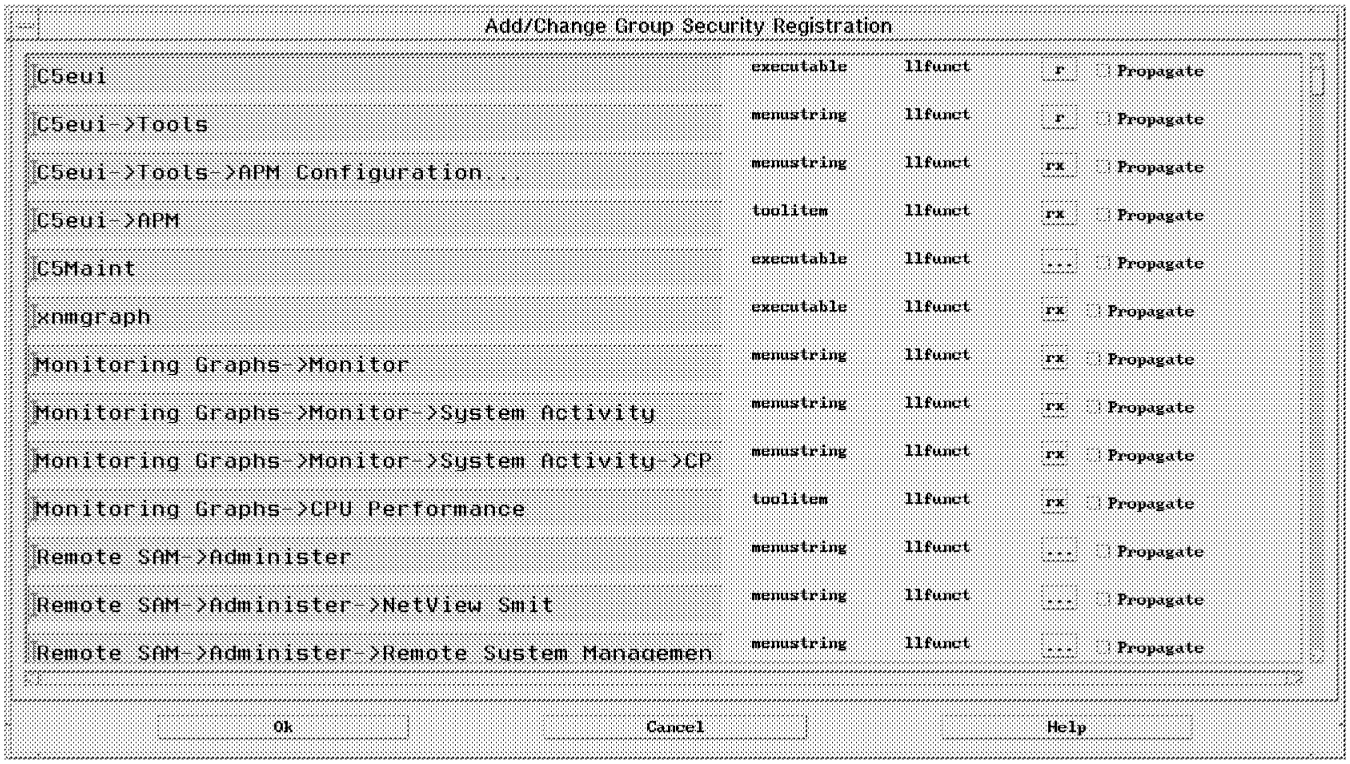


Items	
C5eui	Systems Management Application for Systems Management Resources
NNM-HPUX.gph	Monitors and Graphs CPU Loads
NNM-HPUX.rsam	Remote System Administration Manager
NNM-HPUX.tbl	Obtains and Displays Disk Space
NNM-IP.gph	Monitors and Graphs IP Activity
NNM-IP.tbl	IP Monitoring - Tables
OVWCmds	General NetView commands
PhyAddr	SNMP MIB Monitoring Application
SysInfo	SNMP MIB Monitoring Application
backup	Configuration and Control for Automated Backup
browse.gph	Graphs Collected SNMP Data
collectioned	Collection Facility
correlation	Graphical Logic Rule drawing tool for use with ESE rule processor.
demandpoll	Queries the selected nodes for current information.
filtered	Editor for SNMP Trap Filters
mailtool	Graphical Interface to Mail with capability of sending images.
nmpolling	Configures network monitor (netmon) polling intervals.
nvauth	NetView login, logout, and various other security functions.
nvela	Browsing and Managing the Event Log.
nvevents	SNMP Event Notification Viewer
nvsec_admin	Security Administration Functions
openmon	Discovers and loads the network topology into the GTM database.
ovw	Main User Interface, Submap Windows and Function Menus
printtool	For Capturing Window Images and Printing Files
shpmon	Monitors the local workstation's system.
tralertdfc	Trap-to-Alert Filter Control
xnmbrowser	Browser for SNMP MIB Data
xnmbuilder	MIB Application Builder for SNMP MIB Data
xnmcollect	Defines and controlls SNMP Data Collectors and Threshold Monitors
xnmfault	Displays the failing resources of a selected node.
xnmloadmib	MIB Compiler/Loader for SNMP
xnmreport	Report Generation Functions
xnmsnmpconf	SNMP Community Name and Timeout Configuration
xnmtrap	Event Notification Message & Action, editor for SNMP Trap Configurations
xxmap	Maintains the submaps for Open Topology protocols.

Figure 13. List of SRFs

Although the title of the panel is Select Applications, the entries listed are really the Security Registration Files (SRFs) that define the programs and menu options that are protected by NetView for AIX security. We describe how to define SRFs in 3.9, "Integrating Your Own Applications with Security" on page 41.

The functions are logically divided in the SRF files by application area, but you can choose to display all possible functions by selecting all the lines. Do this and click on **OK** and then **OK** again. After a few minutes you are presented with the list of NetView for AIX Version 4 menu items and executable programs (see Figure 14 on page 31).


Add/Change Group Security Registration

C5eui	executable	llfunct	r	Propagate
C5eui->Tools	menustring	llfunct	r	Propagate
C5eui->Tools->APM Configuration...	menustring	llfunct	rx	Propagate
C5eui->APM	toolitem	llfunct	rx	Propagate
C5Maint	executable	llfunct	...	Propagate
xnmgraph	executable	llfunct	rx	Propagate
Monitoring Graphs->Monitor	menustring	llfunct	rx	Propagate
Monitoring Graphs->Monitor->System Activity	menustring	llfunct	rx	Propagate
Monitoring Graphs->Monitor->System Activity->CP	menustring	llfunct	rx	Propagate
Monitoring Graphs->CPU Performance	toolitem	llfunct	rx	Propagate
Remote SAM->Administer	menustring	llfunct	...	Propagate
Remote SAM->Administer->NetView Smit	menustring	llfunct	...	Propagate
Remote SAM->Administer->Remote System Managemen	menustring	llfunct	...	Propagate

Figure 14. Add/Change Group Security Registration

There are three types of resource defined in this list:

a. menustring resources control the NetView for AIX menu structure

b. toolitem resources control the contents of the NetView for AIX tool bar

c. executable resources control use of programs that invoke the NetView for AIX security API

The access given to a group member is defined in the attributes field towards the right of the panel. This field has three possible values:

... No access (for a menustring or toolitem entry this means that the option will not appear on the user′s screen)

r Read-only (for a menustring or toolitem entry this means that the option will appear greyed-out, that is, not selectable)

rx Read and Execute (user has full access to the function)

As an example, scroll through the panel to the ovw_binary entry. This represents the NetView for AIX EUI itself, so it needs to be set executable (rx) if the user is to be able to start the EUI at all. If you were to click on the **rx** field and remove the check mark from the x, the user would not be able to run it.

We will leave ovw_binary as executable, but we must still assign the menu options that are to be made available to users in the group.

The lines classified as menustring control the menu hierarchy of the main NetView for AIX menu bar, so if we want to allow the user to use some functions we must flag them rx.

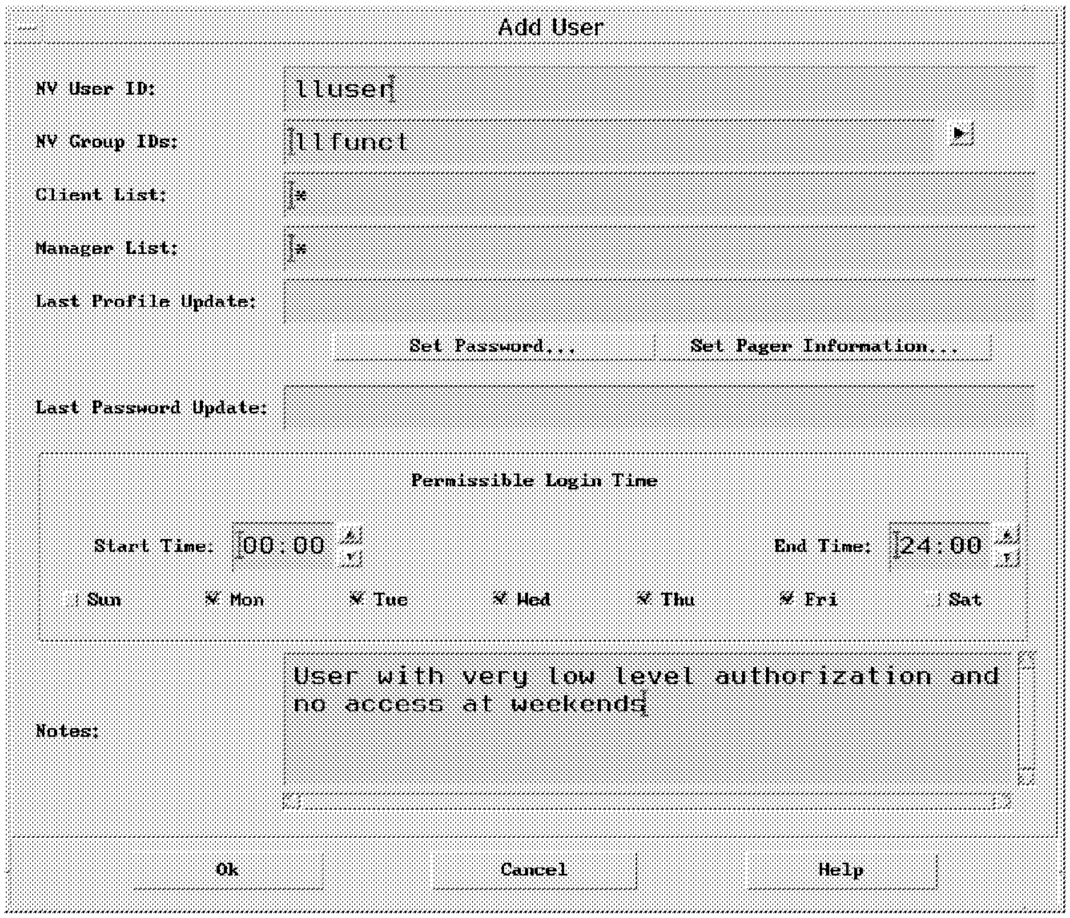
For our low level user we will only allow the File→Exit option and the options under the Locate and Help menus. The Options menu will be shown but will not be executable.

To obtain all these functions you must remove all the r and x indicators from all the lines excluding the ones we are allowing. This is where selecting Propagate can be helpful because it propagates the choices made for the major level to lower levels. For example if we disable access to the NetView Windows->Edit entry and click on Propagate, all the menu entries (Add, Cut, Paste etc.) will also be disabled.

Having defined our llfunct group in this way, we repeated the process thing for the mibit group this time leaving MIB related menu items with the rx option.

7. Add new users and assign them to groups.

   To do this, select **Add** in the Users section to obtain the panel shown in Figure 15.



NV User ID:	lluser
NV Group IDs:	llfunct
Client List:	*
Manager List:	*
Last Profile Update:	
Last Password Update:	

Start Time: 00:00 End Time: 24:00

[ ] Sun [✓ Mon] [✓ Tue] [✓ Wed] [✓ Thu] [✓ Fri] [ ] Sat

Notes: User with very low level authorization and no access at weekends

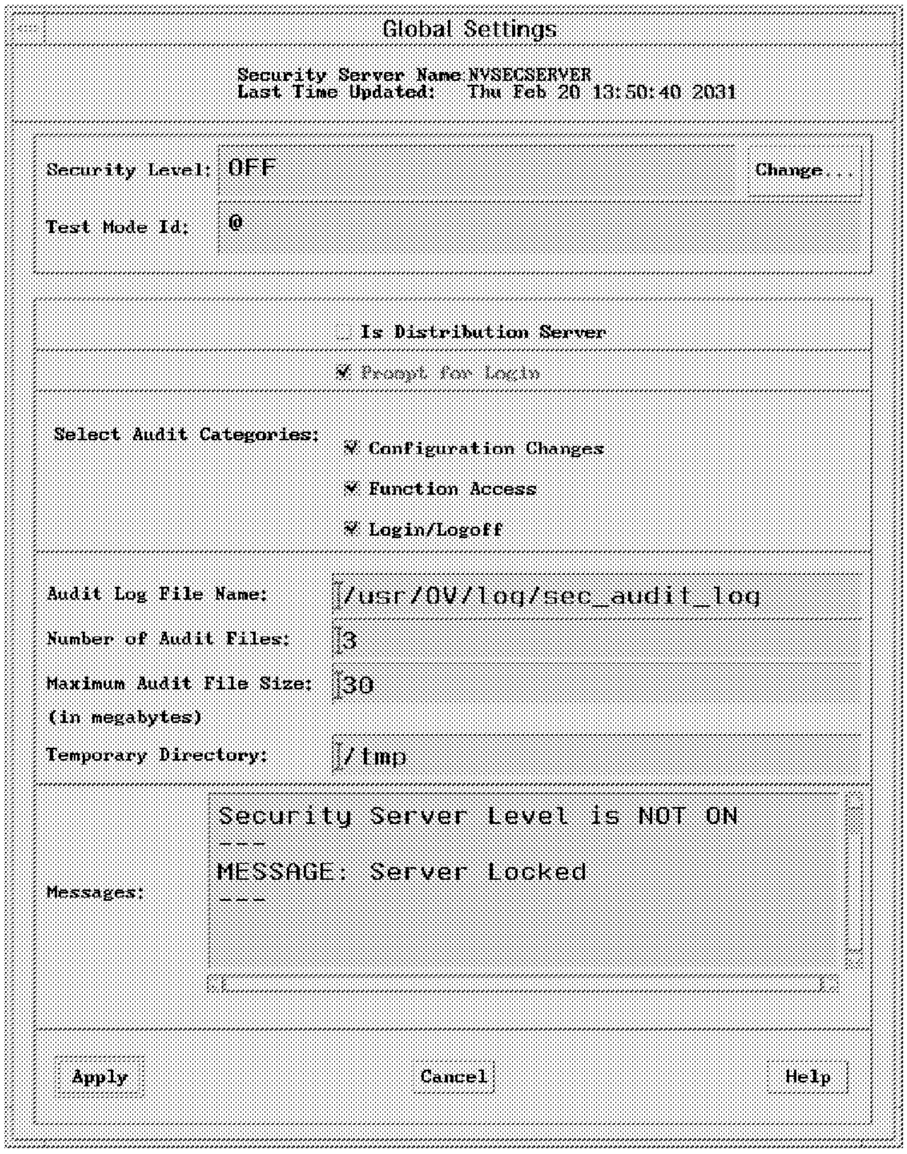
Figure 15. Add User Panel

To create the user ID, simply fill in the fields on the panel. You will notice that in this case we have restricted the user ID to only be usable on weekdays. You have to define a password for the user at this point, by clicking on **Set Password**. If you do not do this, the user will not be able to log in.

After selecting **Ok** the new user ID, lluser, is added to the Users list.

8. Test your security and put it into production.

   Nothing that we have defined so far will have any effect until we activate NetView security at the global level. From the main panel of NetView Security Administration select the Options→Global Settings... from the menu. The Global Settings main panel will be shown (see Figure 16).



Security Server Name NVSECSERVER
Last Time Updated: Thu Feb 20 13:50:40 2031

Security Level: OFF
Test Mode Id: @

Is Distribution Server
✓ Prompt for Login

Select Audit Categories:
✓ Configuration Changes
✓ Function Access
✓ Login/Logoff

Audit Log File Name: /usr/OV/log/sec_audit_log
Number of Audit Files: 3
Maximum Audit File Size: (in megabytes) 30
Temporary Directory: /tmp

Messages:
```
Security Server Level is NOT ON
---
MESSAGE: Server Locked
---
```

Figure 16. Global Settings Main Panel

   From here we selected **Change...** and then **ON** to activate security. We could also have selected the audit options on this panel, but for now we simply clicked on **Apply** to exit the Global Setting panel and activate security (we will discuss the audit options in 3.8, “Auditing” on page 38).

From this point on, all NetView for AIX functions require you to have first authenticated yourself with a user ID and password.

3.5 Login Flow

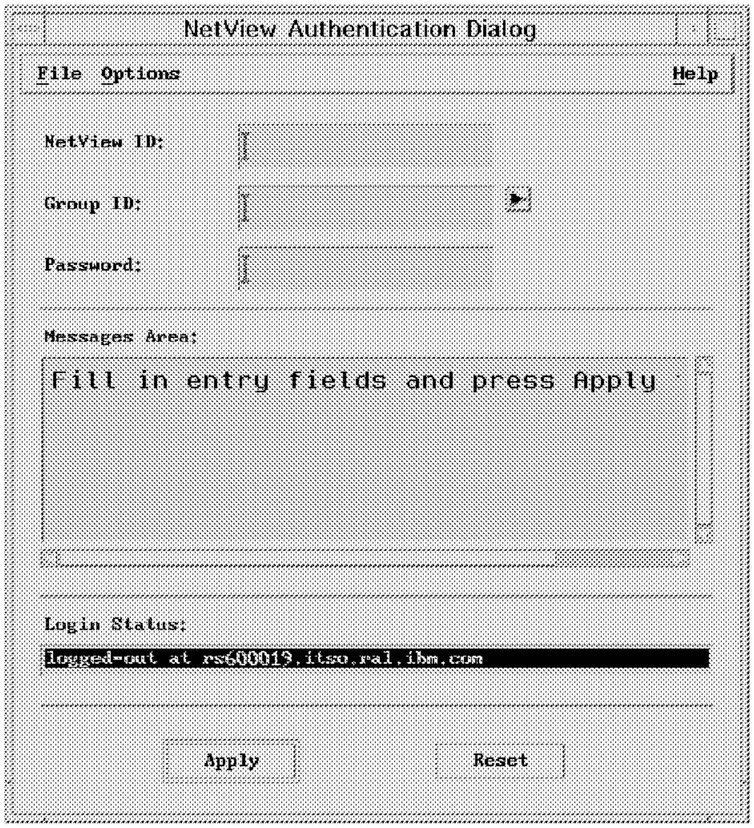
Let us now consider carefully the flow of login operations because now there are two different user validations: one performed by AIX and the one performed by NetView for AIX Version 4.

Suppose that on AIX you have a user called myaixid. With NetView for AIX Version 4 security ON, the complete login flow is as follows:

1. Log in to AIX as myaixid.

   At this point you have the AIX privileges of myaixid but no NetView for AIX Version 4 privileges (that is, you cannot use any NetView command, even if AIX privileges allow you to do it).

2. Start NetView for AIX Version 4. You will see the panel shown in Figure 17.



Fill in entry fields and press Apply

logged-out at rs600019.itso.ral.ibm.com

Figure 17. NetView for AIX Version 4 Login Panel

   Fill the fields with the NetView for AIX user ID, group and password (lluser, llfunct and the password you set in the user definition panel) and click on **Apply**.

   Now you have the AIX privileges of myaixid and the NetView for AIX privileges of llfunct. Because the capabilities of the llfunct group are greatly curtailed, the menu bar will be very sparse (see Figure 18 on page 35).

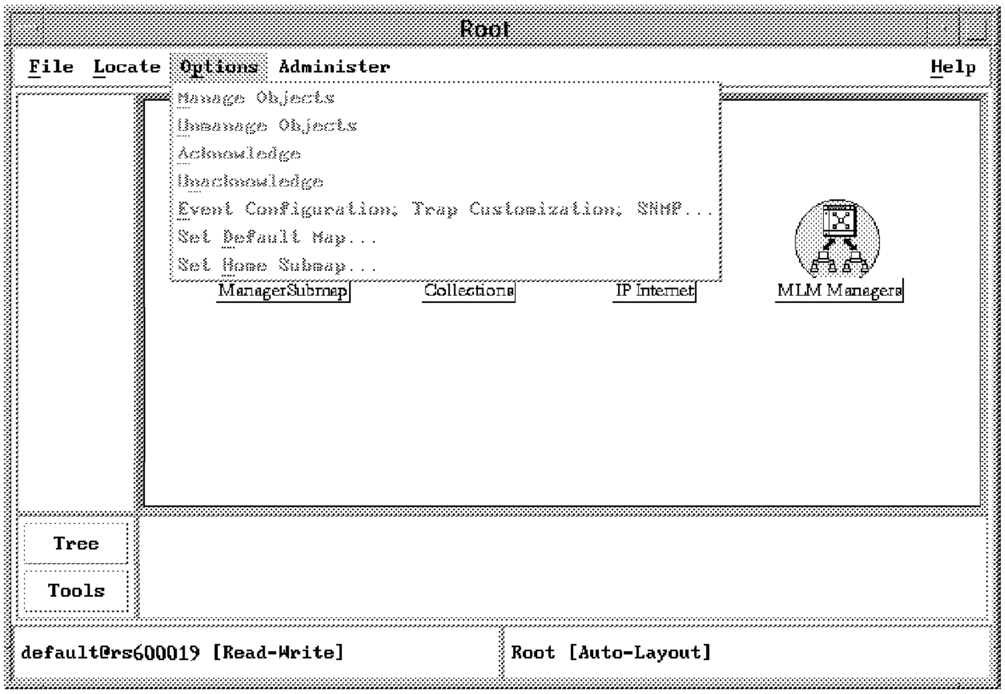


Figure 18. Boring NetView for AIX Version 4 Main Window for User lluser. Not only does this user have a very limited set of menus, but even the ones that are there contain very few selectable entries.

3. Next exit the end user interface and restart it, but this time log in as mibuser in group mibit. Now you have the same AIX privileges but different NetView for AIX privileges.

Warning!

Notice that if you now, from another AIX terminal, log in as myaixid you will find yourself automatically with mibuser NetView for AIX privileges without the need of any kind of NetView password.

This is due to the fact that only one NetView for AIX Version 4 login is allowed for each AIX login.

Each AIX login has a well defined set of AIX functions available. When the user logs in to NetView for AIX, he receives a well defined set of NetView for AIX functions. These two sets of functions are related to the pairing of AIX user ID and NetView for AIX user ID and not related to the terminal from which you are working.

Consider this very carefully when sharing AIX logins or, better, don′t share them when you are using NetView for AIX security features.

A partial solution to this last problem could be the use of a procedure that checks for the name of the user ID during AIX login and prevents the user from logging in a second time.

To prevent the second login modify the /etc/security/login.cfg file adding an auth_method like:

```
CHECK_UNIQUE:
    program = /etc/security/check_unique
```

Then create /etc/security/check_unique with the following statements:

```
who | grep $* >/dev/null 2>&1 && exit 1
exit 0
```

To make NetView for AIX Version 4 login faster you can add the following two statements to your $HOME/.profile file:

```
export NVID=your_netview_id
export NVGID=your_netview_group
```

These two environment variables set the default NetView for AIX Version 4 user ID and group. Of course the password must be given interactively.

Notice also that `ovstart` and `ovstop` commands are *not* protected by any SRF, but you have to be the AIX root user to use them to start and stop NetView for AIX daemons. Note, however, that you can add your own SRF files to protect these commands if you don′t want to give root this possibility.

If you want to login to NetView for AIX without starting the end user interface (to use command line operations for example) just type `nvauth` from the AIX command line. You can use the same application to log out when you have finished. If you are not working from a graphic terminal you can use the following syntax:

```
nvauth -login youruserid yourgroupid
```

and you will be prompted for a password.

Table 2 on page 37 summarizes login/logout flow in various situations:

Table 2. Summary of Login/Logout Flow

Situation	How to login	How to logout
Using EUI	Start `nv6000`	Exit EUI
Using command line commands from a graphic terminal	Start `nvauth`	Start `nvauth`
Using command line commands from a text only terminal	Use `nvauth -login your_userid your_groupid`	Use `nvauth -logout your_userid`

3.6 Major Customization of Menu Bar Options

It may be that you are not too concerned about the ultimate security of your system, but want to tailor users′ NetView EUIs to suit the jobs they do. NetView for AIX Version 4 security provides a consistent and practical method to achieve such major customization on menu bars, tools and context menus. The example shown in the previous section (Figure 18 on page 35) showed how a much simplified menu configuration can be created.

You could do this in previous releases by setting the $OVREGDIR environment variable to point to a customized registration directory. This is a more complex process to set up and administer than defining groups and setting access flags in NetView for AIX Version 4 security, however.

3.7 Shift Takeover

One unique feature of the security mechanism is the shift-out/shift-in capability. What is the difference between logout/login and shift-out/shift-in?

When you log out from NetView for AIX, the end user interface is closed and you start your operations again with a new user ID. With shift-out/shift-in, the EUI is never closed, but the incoming user has to authenticate him or herself to access it. Both activities can leave a track on the audit trail if Login/Logout recording is enabled. The advantage of shift-out/shift-in is that you do not have to suffer the delay of restarting the EUI.

This is the flow of operations for shift-out/shift-in:

1. The departing operator selects: Tools→User Security→Options→Shift_Out to lock the the display.

   Now the screen is locked and there is a big key in a box on the display.

2. The incoming operator clicks on the key box on the screen to obtain the NetView Authentication dialog panel (see Figure 17 on page 34).

3. The incoming operator enters the new NetView for AIX Version 4 user ID and resumes working at the point left by the previous operator.

   The only restriction to this feature is that both operators must belong to the same user group (because menu configuration takes place as part of the EUI initialization process).

3.8 Auditing

Security is nothing without effective logging. NetView for AIX Version 4 security features now include a complete set of auditing functions. A variety of actions are recorded in the audit log. If you activate the default functions you will find the following record types there:

Configuration changes These record specific activities that alter the way that NetView for AIX operates, specifically:

- Event Configuration (updates to trapd.conf)
- Changes to polling intervals
- SNMP configuration (changes to community names and status polling)

Function accesses Creates a record each time a user selects a protected function, whether it is a program or a menu entry.

Login/Logoff log Records all logins, logouts, shift-ins and shift-outs.

3.8.1 Configuring the Audit Options

To configure the audit options, enter the command `nvauth` to go to the main security administration panel and select **Options**→**Global Settings...** from the menu to get to the Global Setting panel (see Figure 16 on page 33).

It is then a simple matter to select which of the three audit categories to activate. The default configuration will be to record everything.

You should also think about whether you want to alter the Audit Log File Name field, you may consider putting it in a different place for security or disk space considerations.

The maximum file size and maximum number of files fields define how much data you will be able to store. The default values (3 files of 30MB each) will allow you to keep approximately 400,000 events before the oldest one is overwritten, but make sure there is enough free space in the file system, particularly if you take the default file name, since if /usr becomes full serious problems can result.

3.8.2 Viewing the Audit Logs

The audit logs are just ASCII flat files, so you can easily look at the contents using standard Unix editors and utilities. Alternatively, you can use the reporting facility that is built in to NetView for AIX.

From the NetView Security Administration Panel (Figure 10 on page 28) select **Options**→**Audit Report** from the menu bar to see the NetView Security Audit panel (see Figure 19 on page 39).

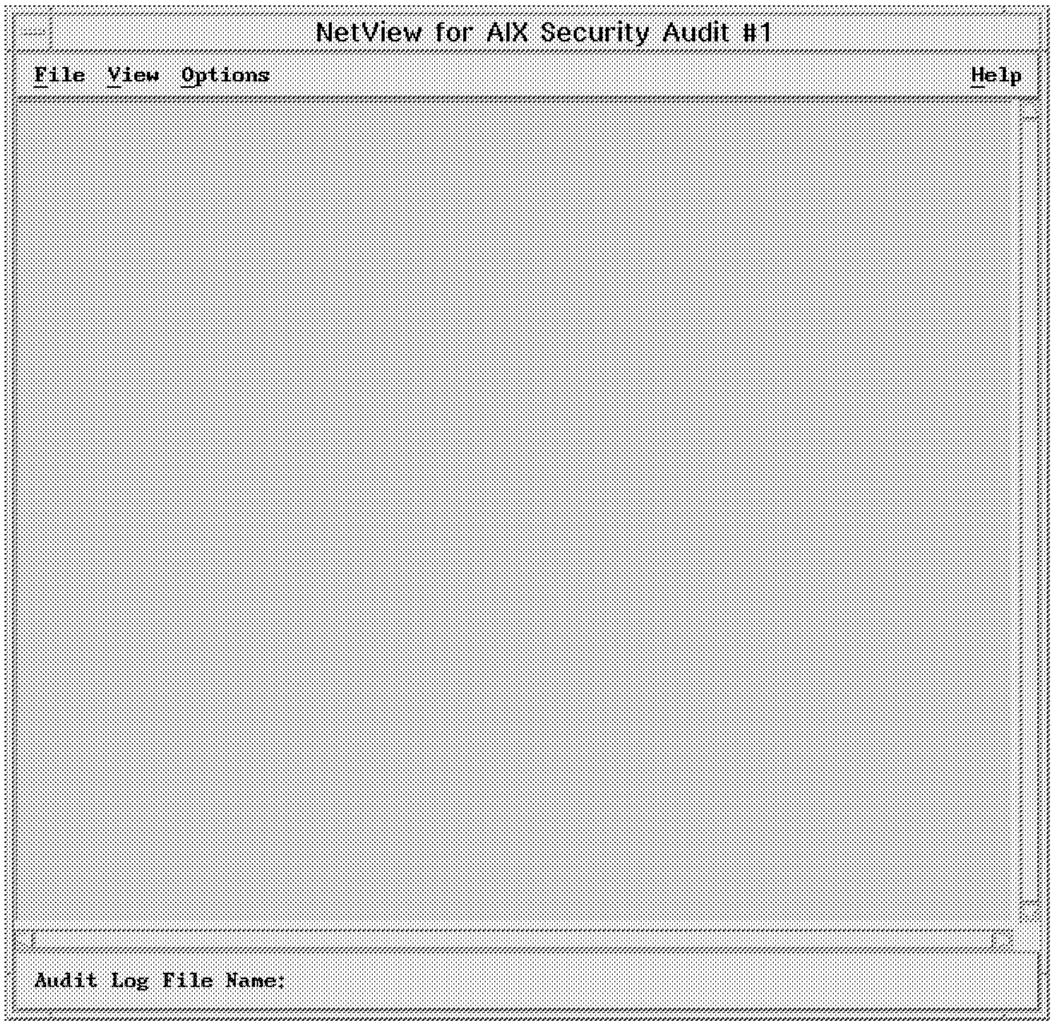
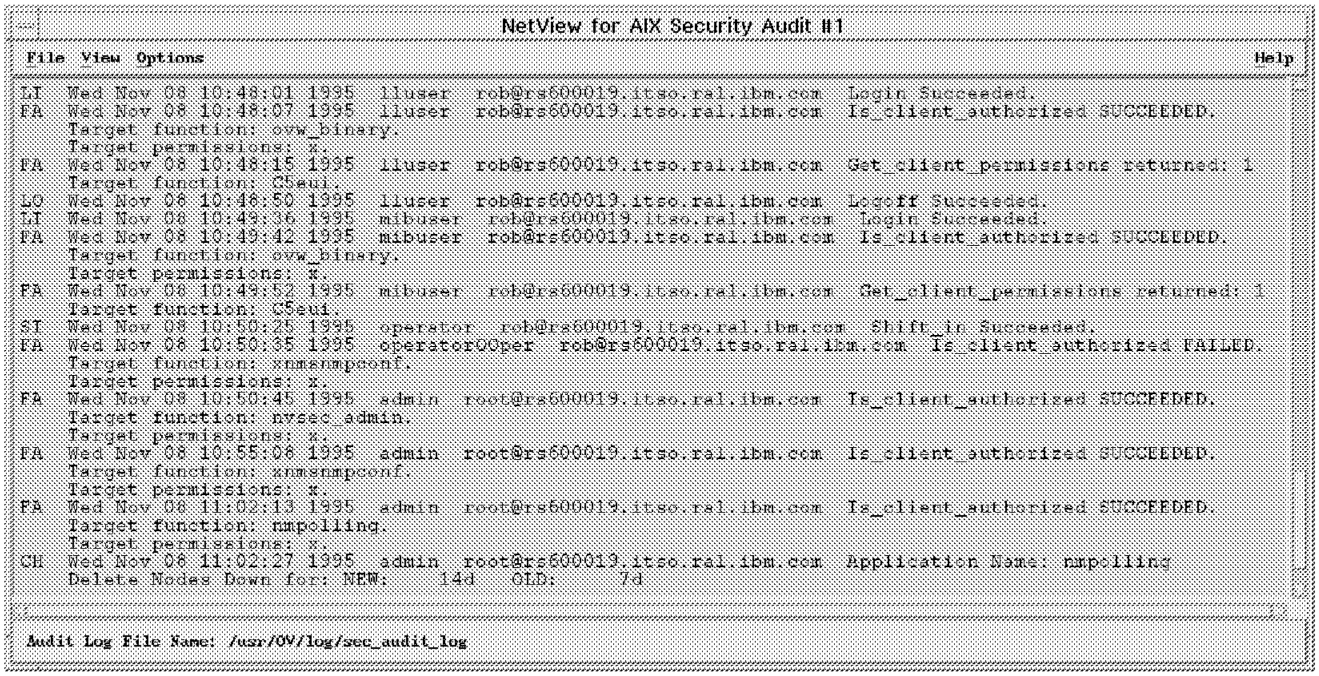


Figure 19. Audit Log Main Panel. This is not very exciting until you select a file to open.

Now select **File**→**Open** and select the audit log from the list to display the activities your system has recorded. Your panel will look something like Figure 20 on page 40.



```
LI  Wed Nov 08 10:48:01 1995  lluser  rob@rs600019.itso.ral.ibm.com  Login Succeeded.
FA  Wed Nov 08 10:48:07 1995  lluser  rob@rs600019.itso.ral.ibm.com  Is_client_authorized SUCCEEDED.
    Target function: ovw_binary.
    Target permissions: x.
FA  Wed Nov 08 10:48:15 1995  lluser  rob@rs600019.itso.ral.ibm.com  Get_client_permissions returned: 1
    Target function: C5eui.
LO  Wed Nov 08 10:48:50 1995  lluser  rob@rs600019.itso.ral.ibm.com  Logoff Succeeded.
LI  Wed Nov 08 10:49:36 1995  mibuser  rob@rs600019.itso.ral.ibm.com  Login Succeeded.
FA  Wed Nov 08 10:49:42 1995  mibuser  rob@rs600019.itso.ral.ibm.com  Is_client_authorized SUCCEEDED.
    Target function: ovw_binary.
    Target permissions: x.
FA  Wed Nov 08 10:49:52 1995  mibuser  rob@rs600019.itso.ral.ibm.com  Get_client_permissions returned: 1
    Target function: C5eui.
SI  Wed Nov 08 10:50:25 1995  operator  rob@rs600019.itso.ral.ibm.com  Shift_in Succeeded.
FA  Wed Nov 08 10:50:35 1995  operatorOOper  rob@rs600019.itso.ral.ibm.com  Is_client_authorized FAILED.
    Target function: xnmsnmpconf.
    Target permissions: x.
FA  Wed Nov 08 10:50:45 1995  admin  root@rs600019.itso.ral.ibm.com  Is_client_authorized SUCCEEDED.
    Target function: nvsec_admin.
    Target permissions: x.
FA  Wed Nov 08 10:55:08 1995  admin  root@rs600019.itso.ral.ibm.com  Is_client_authorized SUCCEEDED.
    Target function: xnmsnmpconf.
    Target permissions: x.
FA  Wed Nov 08 11:02:13 1995  admin  root@rs600019.itso.ral.ibm.com  Is_client_authorized SUCCEEDED.
    Target function: nmpolling.
    Target permissions: x.
CH  Wed Nov 08 11:02:27 1995  admin  root@rs600019.itso.ral.ibm.com  Application Name: nmpolling
    Delete Nodes Down for: NEW:     14d    OLD:      7d
```

Audit Log File Name: /usr/OV/log/sec_audit_log

Figure 20. Displaying Audit Log Entries. In this case we can see examples of login, logout and shift-in records, as well as records from successful and unsuccessful command and menu selection attempts.

The meaning of the different audit log entries are:

LI/LO	These entries identify Login/Logout activities either succeeded of failed, with a timestamp on it.
SI/SO	These entries identify Shift-In/Shift-Out activities.
FA	These entries identify Functional Activities, which means mainly the starting of NetView for AIX Version 4 registered programs like ovw_binary (main EUI) or nvsec_admin and access to menu items and functions.
CH	These entries identify Configuration Changes (like changing polling intervals in our sample case).

You can filter the view of the audit log by setting criteria for text strings, user IDs and record categories. You do this by choosing **View→Set Criteria** from the menu bar and filling in the View Criteria panel (see Figure 21 on page 41).

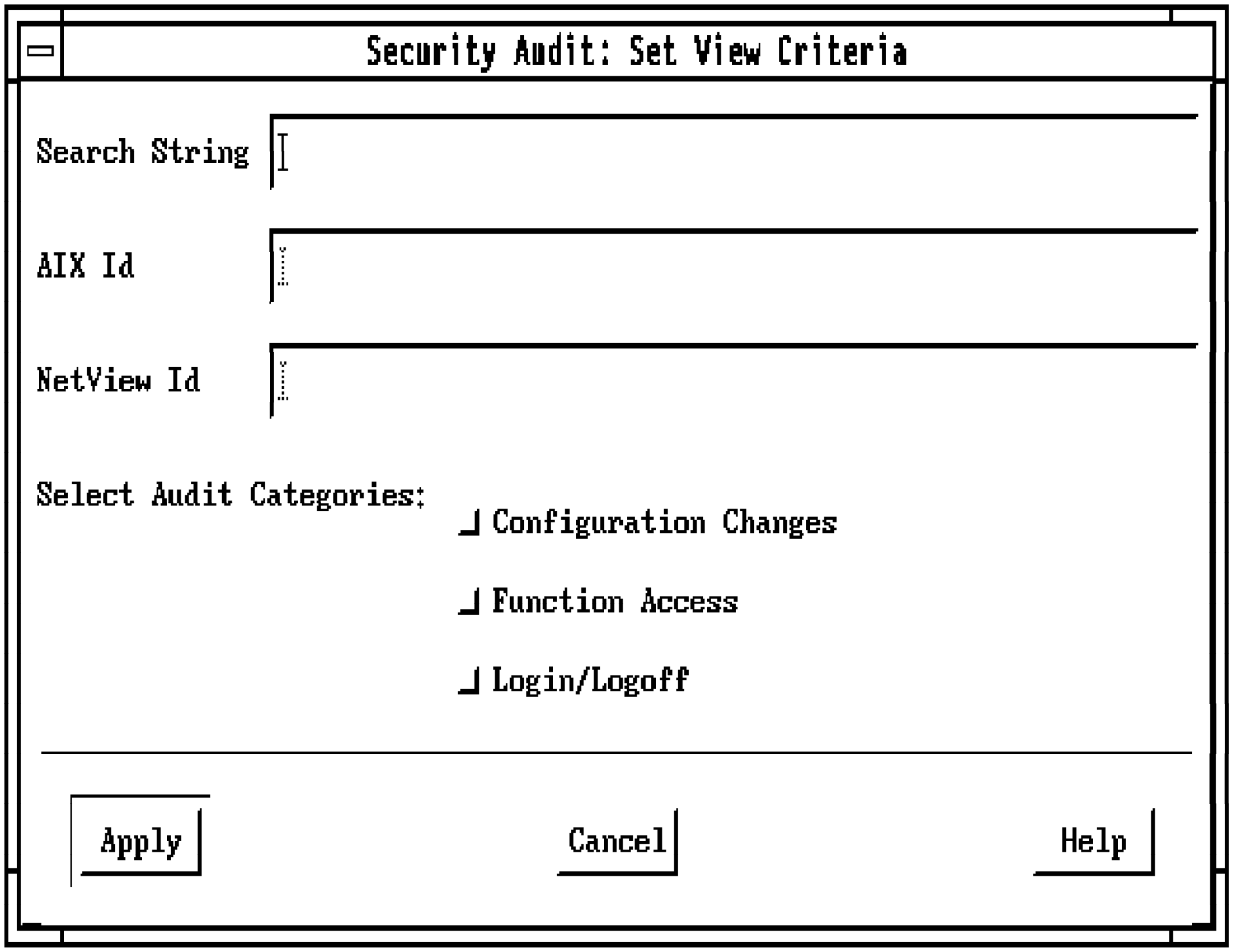


Figure 21. Set Audit View Criteria

3.9 Integrating Your Own Applications with Security

If you have an application which integrates in some way with NetView for AIX Version 4, you can choose to add it into the NetView for AIX security configuration.

The following are the steps to do this:

1. Assess which elements to protect
2. Build a Security Registration File (SRF)
3. Optionally, modify the Application Registration File (ARF) of your application
4. Optionally, modify the code of your application to add calls to the security APIs

In many cases you will only take the first two actions. These give control over applications that are launched from the menu bar and tool panel. Of course this will not prevent someone executing your application from the command line, outside of NetView for AIX, but you may consider that this is enough protection. Often the requirement for security is to prevent users from doing foolish things accidentally, not to repel a determined hacker.

You only need to take the last action (modify the code) if your application can be started outside the NetView for AIX end user interface and you want thorough protection or if it needs to explicitly access security functions (like auditing for example).

You should bear in mind that if you don′t implement an SRF for your application and NetView for AIX security is active, your application can be run by anybody, and any menu entries it creates will appear on all user′s displays. This means you have to be thorough when creating SRFs if you are employing the security mechanisms to do menu personalization (see 3.6, "Major Customization of Menu Bar Options" on page 37).

You also need to consider whether your application will conflict with NetView for AIX functions. For example, your application may be a shell script that invokes the `ovobjprint` command to get object database field information. ovobjprint is protected by an SRF, so any users that are authorized to use your application will also need to be authorized to execute ovobjprint.

We will now look at each of the steps in a little more detail.

3.9.1 Determine What You Need to Secure

Consider:

1. Menu bar items
2. Object context menu items
3. Tool window items
4. Command line commands
5. Configurable resources (these can be audited)

3.9.2 Build an SRF File

SRF files are located in /usr/OV/security/$LANG/Domains/registration. You will avoid confusion if you call the SRF by the same name as the corresponding ARF file.

For more information about the syntax used in SRF files see *NetView for AIX Version 4 Programmer′s Guide* SC31-8164.

In general all the information needed for your SRF files are in the ARF files so you can create them automatically by processing the ARF definition with the `c_arf2srf` command.

Warning!

Pay a lot of attention to special characters (like square brackets, braces) when you are creating SRFs from ARFs. If they are left inside the SRF file they may cause problems for nvsec_admin during syntax checking.

3.9.3 Modify the ARF

Application Registration Files (ARFs) are in the /usr/OV/registration/C directory and its subdirectories. You may want to modify them to suit your security needs.

In the Action stanza, you can add the keyword Security to specify that this action is *only* available if NetView for AIX Version 4 security is active (ON).

For more information about ARF files see *NetView for AIX Version 4 Programmer′s Guide* SC31-8164.

3.9.4 Code New API Calls

Remember that only if your application needs to be protected from access *outside* the NetView for AIX end user interface, must you modify your code.

The following are the new API calls related to security included in NetView for AIX Version 4:

nvs_isClientAuthorized()	Code this at the start of the program. It returns SEC_AUTHORIZATION_SUCCESS if the client does have the required access and SEC_AUTHORIZATION_FAILURE otherwise. You can also use this API to check permissions on the other elements of your application, such as menu items, files, etc.
nvs_getClientPerms()	Code this for more detailed information on the permissions that the user has on an element (read only, read write, etc.).
nvs_Audit()	Use this only if you are interested in writing your own messages in the audit log.
nvs_deleteSecContext()	Code this just before your application terminates.

The security APIs require you to add #include statements for the following header files:

```
/usr/OV/include/OV/sec_api.h
/usr/OV/include/OV/sec_errs.h
```

3.9.5 A Sample Integration

In this chapter we′ll see an example of this integration.

We chose for our example a program called wteuiap4.Figure 22 on page 44 shows the program README file.

```
1. login to the AIX as root user
2. Copy files Submap_List and Xpsp to the / directory
3. Copy file wteuiap4.reg to the /usr/OV/registration/C directory
4. Start NetView for AIX from smit or issue the
   /usr/OV/bin/nv6000 command, and
   the new item Wteuiap4 will appear in the menu bar.
5. You can test the program by selecting Wteuiap4 from the menu
   bar in NetView for AIX.
6. The first menu item ″Submap List″ will list the IDs and names of
   all submaps in a scrollable window
7. The second menu item ″Process Monitor″ will add an executable icon inthe
   the Root submap.  The user can double click the icon to start the
   process tree program.                                                only
```

Figure 22. Wteuiap4 Readme File

As you can see from the README, this application adds some menu entries to NetView for AIX. They are defined by the registration file shown in Figure 23.

```
/*
Registration for OVw API sample application WTEUIAP4
*/

Application ″OVw API Example WTEUIAP4″ {

            Description  {
                ″Sample API menu program″
                 }

// wteuiap4 resided in the path /wteuiap4/wteuiap4
// it can be initiated from NV/6K menu bar

MenuBar ″Wteuiap4″ _W {
″Submap List″ f.action Lsubmap;
″Process Monitor″ f.action Picon;
}
Action Lsubmap {
Command ″/wteuiap4/wteuiap4″;
}
Action Picon {
Command ″/wteuiap4/addicon″;
}
Action Xpsp {
Command ″/wteuiap4/xpsp -zoomed &″;
}
}
```

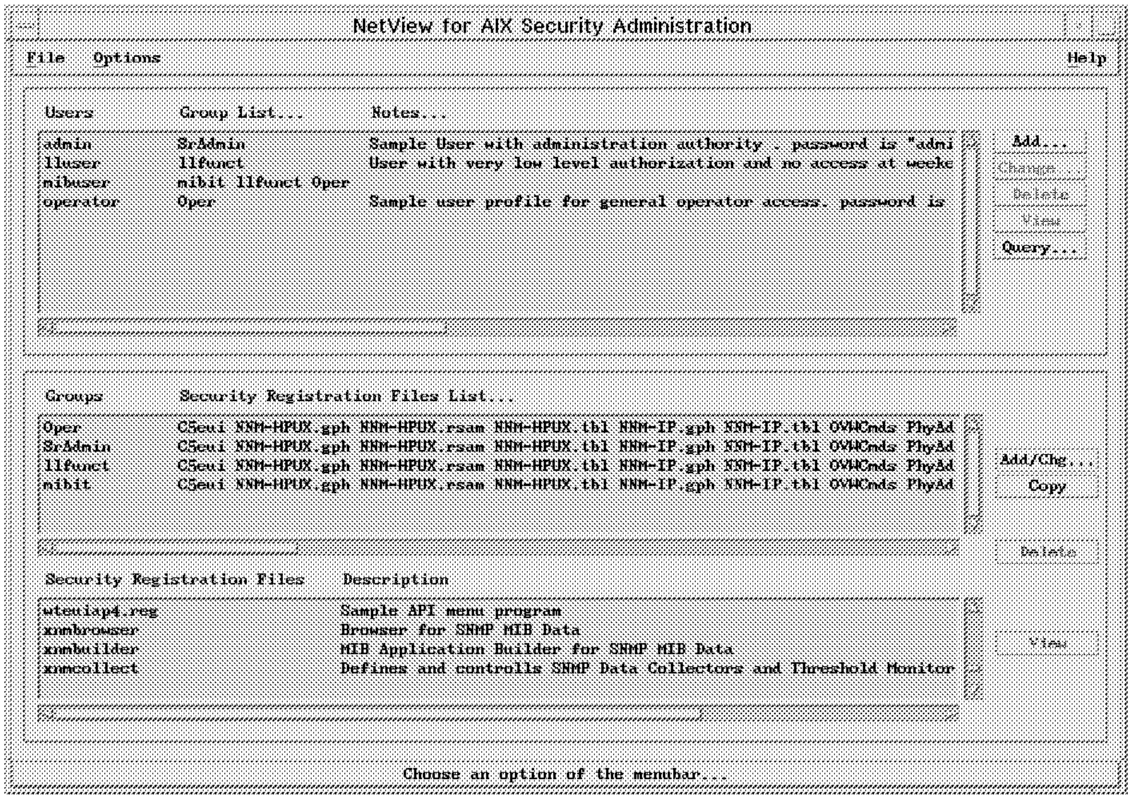
Figure 23. Wteuiap4 Registration File

We took this registration file and entered the command `c_arf2srf wteuiap4.reg` to create a SRF from it. Figure 24 shows the resulting file.

```
#
DOMAIN_ID = WTEUIAP4
DESCRIPTION = ″Sample API menu program″
SEPARATORS = ->
VALID_PERMISSIONS =  rx
ELEMENTS =
″OVw API Example WTEUIAP4->Wteuiap4″ . FALSE menustring
″OVw API Example WTEUIAP4->Wteuiap4->Submap List″ . FALSE menustring
″OVw API Example WTEUIAP4->Wteuiap4->Process Monitor″ . FALSE menustring
#
```

Figure 24. Wteuiap4 SRF File

The program creates the SRF with the same name as the original ARF and places it in directory /usr/OV/security/$LANG/Domains/registration. Thereafter, when we enter security administration (using the `nvsec_admin` command or the option from the Administer menu) we see the new application added to the list in the Administration Panel (see Figure 25 on page 45).


NetView for AIX Security Administration

Users	Group List...	Notes...
admin	SrAdmin	Sample User with administration authority . password is "admi
lluser	llfunct	User with very low level authorization and no access at weeke
mibuser	mibit llfunct Oper	
operator	Oper	Sample user profile for general operator access. password is

Groups	Security Registration Files List...
Oper	C5eui NNM-HPUX.gph NNM-HPUX.rsam NNM-HPUX.tbl NNM-IP.gph NNM-IP.tbl OVWCmds PhyAd
SrAdmin	C5eui NNM-HPUX.gph NNM-HPUX.rsam NNM-HPUX.tbl NNM-IP.gph NNM-IP.tbl OVWCmds PhyAd
llfunct	C5eui NNM-HPUX.gph NNM-HPUX.rsam NNM-HPUX.tbl NNM-IP.gph NNM-IP.tbl OVWCmds PhyAd
mibit	C5eui NNM-HPUX.gph NNM-HPUX.rsam NNM-HPUX.tbl NNM-IP.gph NNM-IP.tbl OVWCmds PhyAd

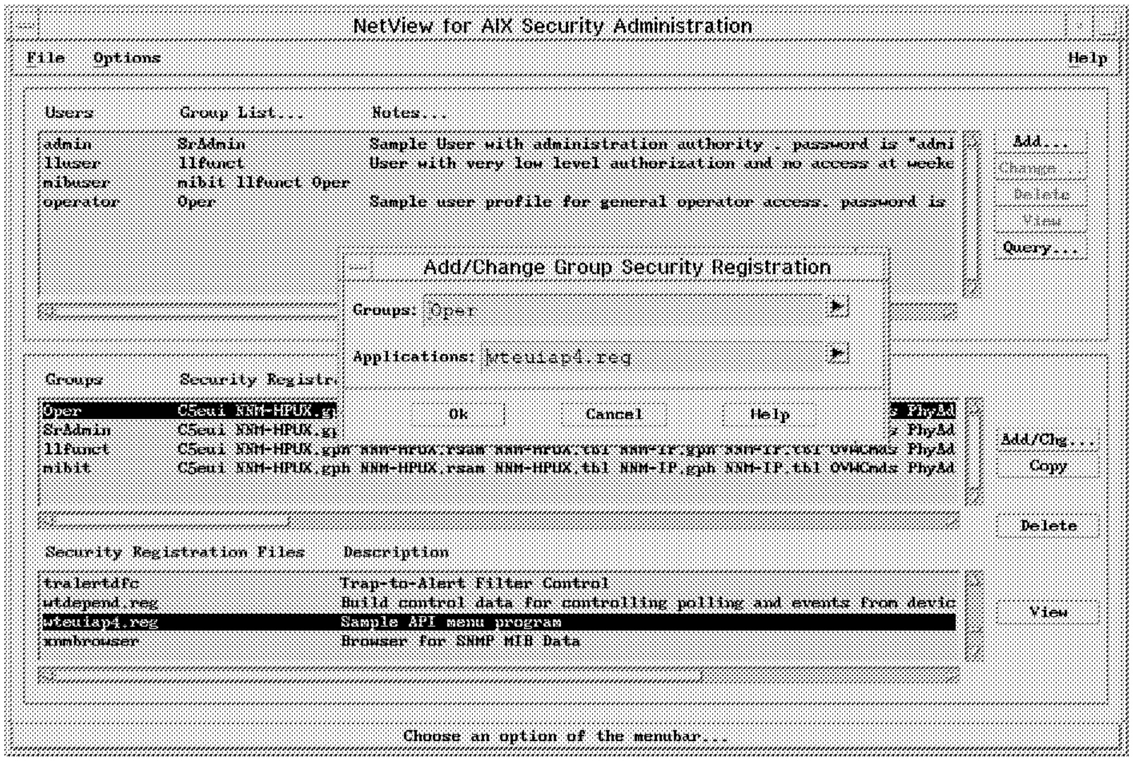
Security Registration Files	Description
wteuiap4.reg	Sample API menu program
xnmbrowser	Browser for SNMP MIB Data
xnmbuilder	MIB Application Builder for SNMP MIB Data
xnmcollect	Defines and controlls SNMP Data Collectors and Threshold Monitor

Choose an option of the menubar...

Figure 25. Security Administration Panel with wteuiap4.srf

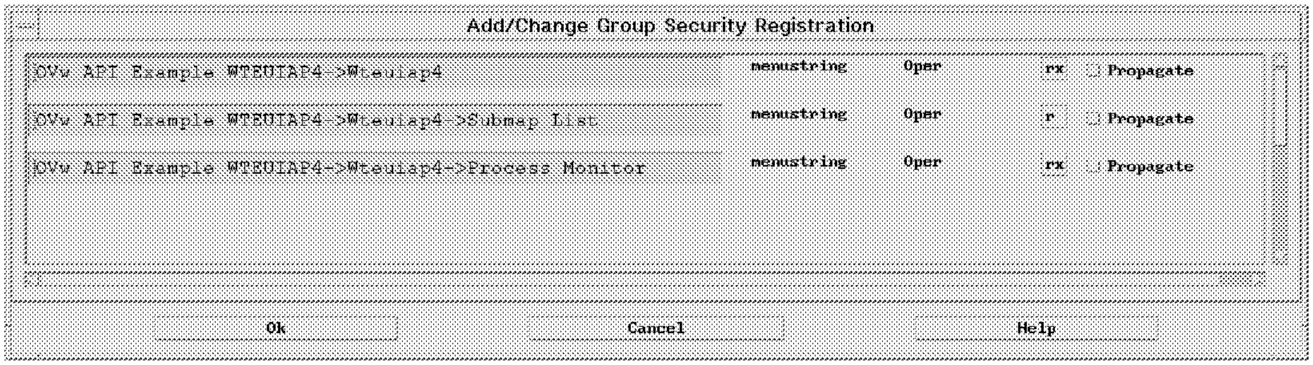
Now we can add permissions for the menu entries defined in the SRF to user groups, as described in 3.4, “How to Create a Trusted Environment” on page 26. We assigned full (rx) permission to the SrAdmin group but only allowed the Oper group to be able to execute the Submap List option.

The sequence of configuration screens for the Oper group is shown in Figure 26 on page 46 and Figure 27 on page 46.



```
NetView for AIX Security Administration
File   Options                                                          Help

Users         Group List...        Notes...
admin         SrAdmin              Sample User with administration authority . password is "admi
lluser        llfunct              User with very low level authorization and no access at weeke
mibuser       mibit llfunct Oper
operator      Oper                 Sample user profile for general operator access. password is

                    Add/Change Group Security Registration
                    Groups: Oper
                    Applications: wteuiap4.reg
                       Ok          Cancel          Help

Groups        Security Registr...
Oper          C5eui NNM-HPUX.g                                                                 PhyAd
SrAdmin       C5eui NNM-HPUX.g                                                                 PhyAd
llfunct       C5eui NNM-HPUX.gph NNM-HPUX.rsam NNM-HPUX.tbl NNM-IP.gph NNM-IP.tbl OVWCmds PhyAd
mibit         C5eui NNM-HPUX.gph NNM-HPUX.rsam NNM-HPUX.tbl NNM-IP.gph NNM-IP.tbl OVWCmds PhyAd

Security Registration Files    Description
tralertdfc                     Trap-to-Alert Filter Control
wtdepend.reg                   Build control data for controlling polling and events from devic
wteuiap4.reg                   Sample API menu program
xnmbrowser                     Browser for SNMP MIB Data

Choose an option of the menubar...
```

Figure 26. Selecting Group and SRF to Modify



```
Add/Change Group Security Registration
OVw API Example WTEUIAP4 >Wteuiap4                    menustring   Oper   rx   Propagate
OVw API Example WTEUIAP4 >Wteuiap4 >Submap List       menustring   Oper   r    Propagate
OVw API Example WTEUIAP4->Wteuiap4->Process Monitor   menustring   Oper   rx   Propagate

        Ok                    Cancel                    Help
```

Figure 27. Defining Access Permissions for the Oper Group

If we now log in to NetView for AIX and start the end user interface, we can see two different menu bars depending on the user ID and group used for the connection.

The menu bar for the Oper group with both options present but only one available is shown in Figure 28 on page 47.

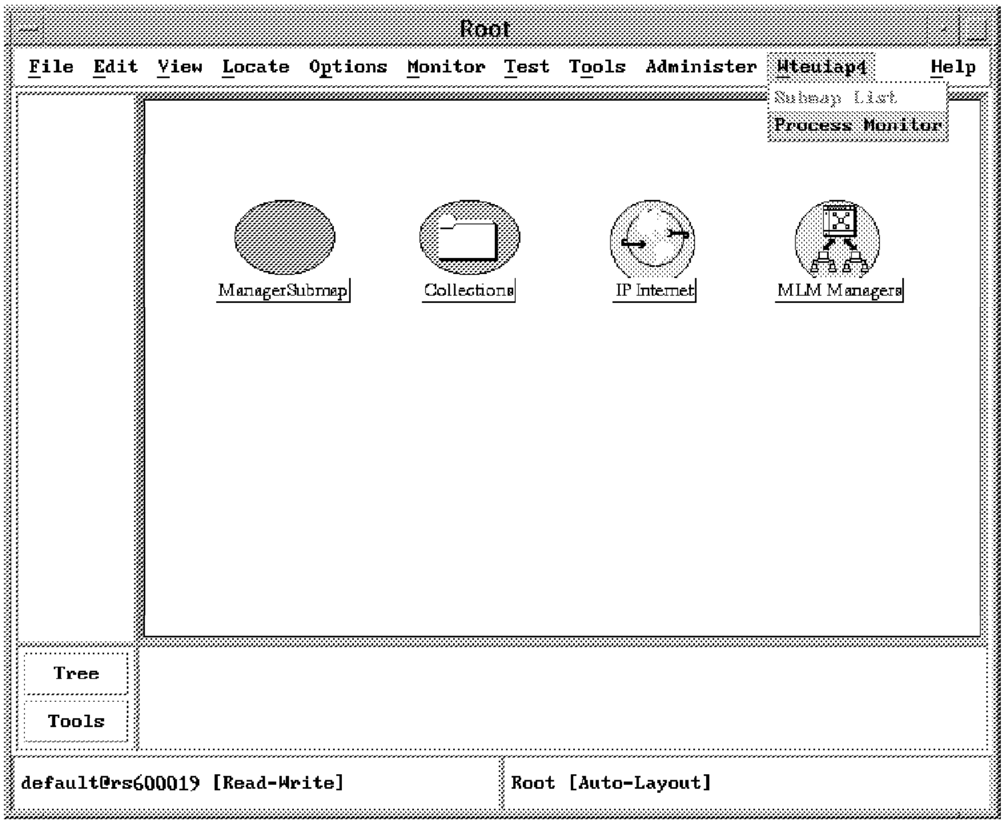


Figure 28. NetView for AIX Version 4 Root Panel for Users in Group Oper

For the Oper group you′ll see the same menu bar but with all options greyed-out in the pull-down menu of Wteuiap4.

If the program can be started by the command line, you should add the security APIs to check authorization. To do that, you should add a line containing the executable name to the SRF:

```
″wteuiap4″ . FALSE executable
```

Click on the **Add/Chg...** button to set permissions related to the Oper and SrAdmin groups. Next, you need to add the API calls to your code. Figure 29 shows a code fragment from wteuiap4 which performs a simple check to see if the user has the execute (x) permission.

```
#include <OV/sec_api.h>
#include <OV/sec_errs.h>

void main (int argc, char ** argv)
{
  int authorized, status;

  /* Check if the user has permissions to execute this program. */
  nvs_isClientAuthorized (″wteuiap4″, ″x″, &authorized, &status);
  if ((status != SEC_SUCCESS) || (!authorized))
    {
      /* Treat error. */
      nvs_deleteSecContext ();
      exit (1);
    }
```

Figure 29. Example of nvs_isClientAuthorized API Call

If you want to get details of the permissions of a protected object (menustring, toolitem or executable), you should encode something like the fragment shown in Figure 30 on page 48.

```
int permissions, status;

nvs_getClientPerms (″WTEUIAP4->Wteuiap4->Submap List″, &permissions,
                    &status);
if (status != SEC_SUCCESS)
  {
    /* Treat error. */
    nvs_deleteSecContext ();
    exit (1);
  }
/* Check if the element has read permission. */
if (permissions & SEC_ACL_PERM_READ)
  /* Take the necessary actions. */
else
  /* Check if it has write permission. */
  if (permissions & SEC_ACL_PERM_WRITE)
    /* Take the necessary actions. */
```

Figure 30. Example Using the nvs_GetClientPerms API Call

Some API calls (for example, nvs_isClientAuthorized) will automatically generate an audit log record if they are unsuccessful. However, you may want to add further description information to the log, or report errors that your code detects internally. The code fragment in Figure 31 shows how to do this.

```
char * auditMsgs[3]; /* 3 messages will logged */

auditMsgs[0] = strdup (″Message 1″);
auditMsgs[1] = strdup (″Message 2″);
auditMsgs[2] = strdup (″Message 3″);

if (nvs_Audit (ATYPE_CONFIG, 3, auditMsgs) != SEC_SUCCESS)
  /* Treat error. */
```

Figure 31. Example Using the nvs_Audit API Call

When compiling programs that use the security APIs, you will also need to link some additional libraries. They are: libnvsec.a and libnvgss.a, located in `/usr/OV/lib`.

3.10 Client/Server Considerations

When operating from a NetView for AIX client EUI *without* using the security feature, there are some menu bar selections that cause the user to be prompted for the root password of the server machine. These selections are all for functions that update server configuration parameters, such as polling intervals, rulesets and event customization. When you implement NetView for AIX security the client user is authenticated to the server when he starts the EUI. This means that the server no longer needs to prompt the user for a password when he selects one of the configuration options. From the user′s point of view this means his life is much simpler, because the behavior of the EUI is identical, whether he is running on the server or a client machine.

These new functions involve two new daemons:

nvsecd	Is the daemon running in the server
nvsecltd	Is the daemon running in the client

If you are working in a client/server environment these two daemons talk through the network using encrypted tokens. If the tokens match the value stored in the server security database, the client is granted the access with the appropriate rules.

This conversation uses the Generic Security Services API interface and can be opened to a third party security manager compliant with this architecture. Remember that in a client/server environment SRF files are located on the server only.

Date/Time

The encryption mechanism used by the client and server daemons uses the current time as a seed to ensure that messages are fresh and to counter a ″man in the middle″ hacking attack. This mean that the system clocks of client and server systems need to be synchronized within a few minutes of each other.

As an example of how the various daemons work together and how they relate see Figure 32.

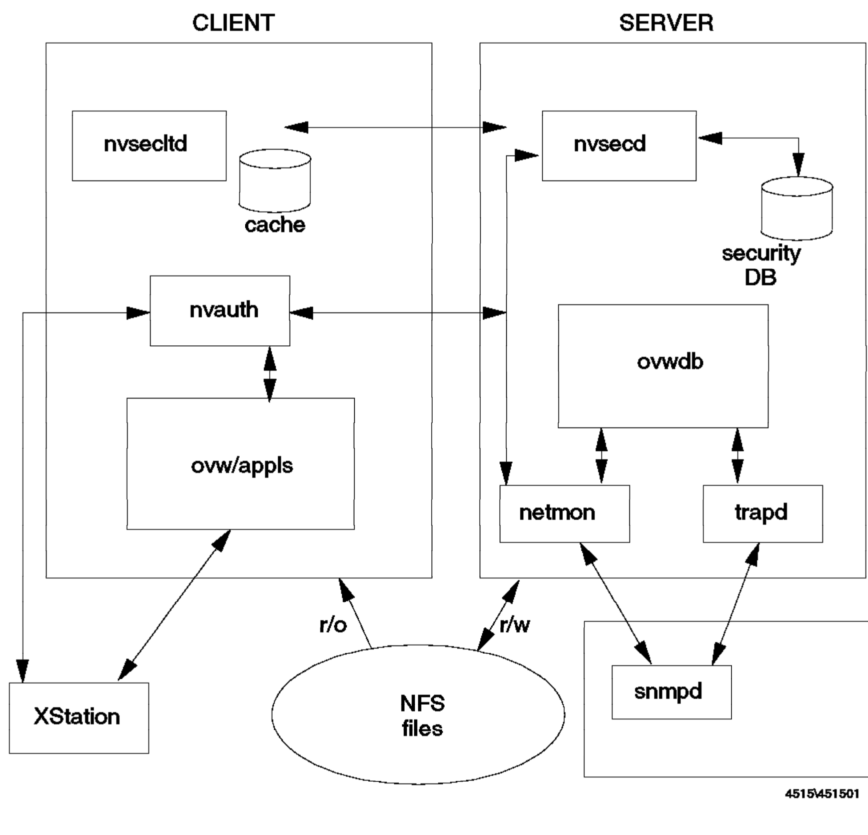


Figure 32. Relationships Among the Security Daemons

From a user perspective nothing changes in a client/server environment. That is to say, all functions and capabilities are identical in either environment.

3.11 Multiple Servers Environment

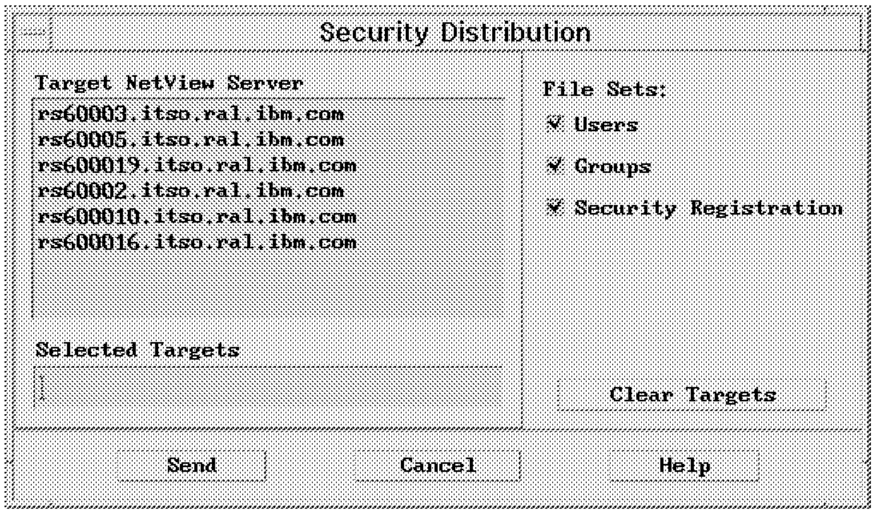
If you are going to implement a multi-NetView for AIX Version 4 Server environment, you must take care of distributing your security configurations to all the servers in the network in such a way to keep all servers updated.

For this eventuality, NetView for AIX Version 4 implements a security distribution feature which gives you the opportunity to distribute security configuration files to multiple servers without the need to configure each one.

To activate the security distribution feature you must define a central distribution server. In this server you must start the NetView Security Administration panel and select **Options**→**Global Settings** from the menu bar.

Now, on the Main distribution machine, select **Is Distribution Server**. When you click on **Apply** you return to the main panel, but now under the Options pull-down menu an entry called Distribution is available.

Select it to see the Security Distribution panel (see Figure 33).


Security Distribution

Target NetView Server

rs60003.itso.ral.ibm.com
rs60005.itso.ral.ibm.com
rs600019.itso.ral.ibm.com
rs60002.itso.ral.ibm.com
rs600010.itso.ral.ibm.com
rs600016.itso.ral.ibm.com

Selected Targets

File Sets:

✓ Users
✓ Groups
✓ Security Registration

Figure 33. Security Distribution Panel

On the target server on which you are going to download the configurations you must stop NetView for AIX and, of course, have the security switch set ON. Now you can select your target server (or servers) and choose what components of the security configuration to download.

If this is the first distribution, you should download everything but after that you only need to download the changes you′ve made. Select **Send** to replicate the configuration files. The new security files will be available in the target server at the next restart of NetView for AIX. The old configuration files in the target server will be saved as /usr/OV/security/$LANG/*.timeddmmyy.

If a manager takeover takes place, the new manager will handle the network using its own security rules, so it is really very important to keep the security configuration distributed and up-to-date in all servers on the network using this method.

3.12 NetView Service Point Considerations

Security impacts the way that NetView Service Point works. When there is an active connection between NetView for AIX and NetView/390 you can issue NetView for AIX Version 4 commands from a host console or MSM graphical user interface using the RUNCMD command.

If you activate security on NetView for AIX all the commands you send from host will fail unless there is a user logged in NetView for AIX.

If no user is logged in to NetView for AIX you′ll receive a message back on the NetView/390 console saying you are not authorized to use NetView for AIX Version 4.

Chapter 4. The Collection Facility

This chapter explains the Object Collection Facility, where this feature can be applied today, and where it could be utilized as a tool for assistance when developing APIs.

The NetView Object Collection Facility can be used to group database objects in separate submaps, perform operations on the set of objects and use all submap resources over them. At collection creation, a collection rule is defined, and it determines which database objects will be part of the collection. This rule can define a subnet, a list of nodes, a common database attribute or another collection rule. The collection is updated dynamically, adding new objects if they match the rule, or deleting them if they do not match it anymore.

4.1 Summary of NetView for AIX V4 Object Collection Facility

The Object Collection Facility is a tool created to help network managers to group database objects in subsets, accordingly to their characteristics. The collections are organized by name, but they follow a definition rule to inquire the database and find which objects will join the collection. A rule uses four types of definitions: node list, subnet, object attribute or another collection rule. All of them can be combined together, using Boolean logic (logical ANDs and ORs), allowing many different rules to be used, one for each case. These definitions are detailed later in this chapter.

Collections can also be used to help in monitoring some kind of object, by putting them in a separate submap. Even better, this new submap is dynamically updated, as objects join or leave the collection and as their status changes. For example, if some object is added to the database and it matches the collection rule, it will be automatically copied to the map, without user intervention.

Collections submaps are located under the Collections icon, displayed on the Root submap. Each collection has a submap and can be seen separately, by double-clicking on the desired icon.

There are two ways to create a collection: the Collection Editor and the Collection APIs. Both tools have enough flexibility to build a collection as complex as desired. Other operations can be performed for a collection such as to modify the collection or delete it.

4.2 NetView for AIX V4 Collection Types

The primitives are used to compose rules that define what kind of objects are in the collection. Some examples on how to use them will be shown in this chapter.

There are four primitive collection types:

Node List	This is a list of all desired nodes, selecting them by their Selection Name or using selected objects from the map.

What′s In a Name?

NetView for AIX has several places in which it keeps names for resources, with the result that something can be known by several different names. The name that ties all these together is the *Selection Name*, which is the key for the entry in the object database. So a resource may be represented in some submaps as an IP name and in others as a MAC address, but all symbols refer back to the one, unique, selection name.

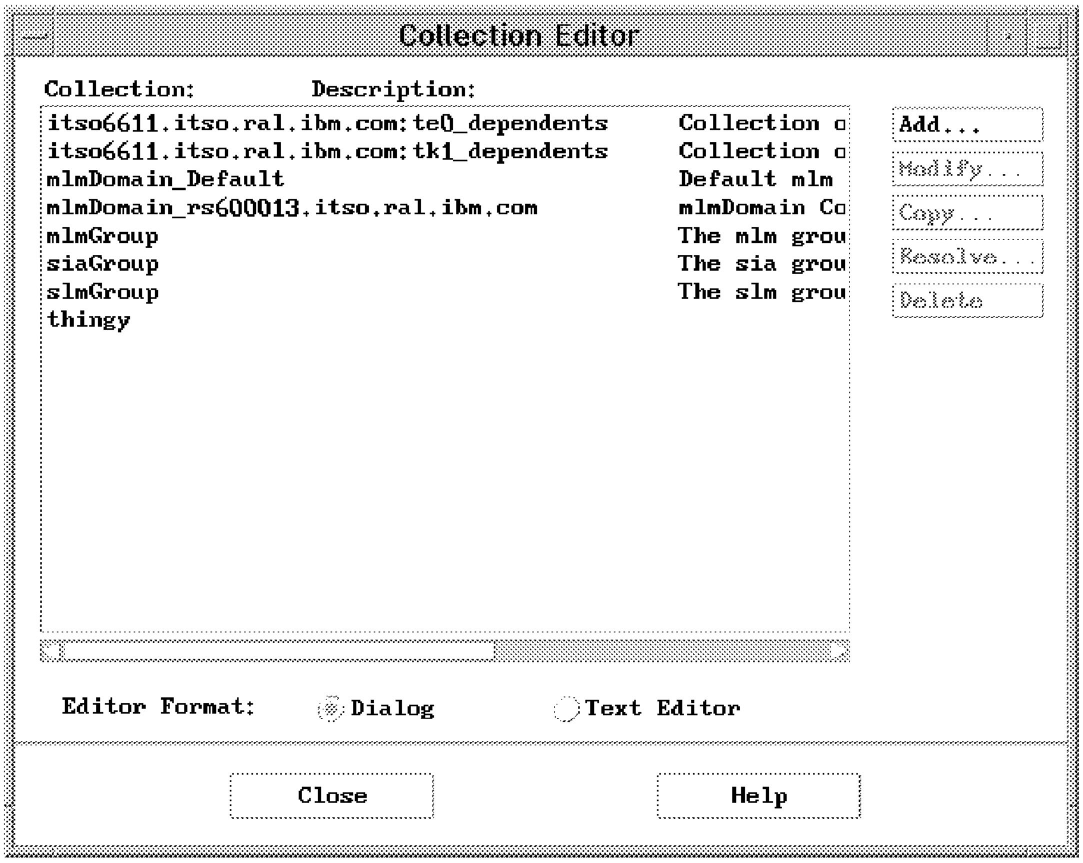
Attribute The attribute is one of the fields that describe the object in the database. The Collection Facility makes a query on all objects in the database and finds which ones have the attribute value set to the value specified. In the Dialog Editor there is a list of all attributes defined in the database, and when an attribute is selected, a list of possible values is displayed.

Collection Rule A previously defined collection rule can be used to be part of a new collection definition. This primitive allows creating more complex rules, based on a set of previously defined rules. For example, a complex rule can be divided on three simple rules. They are tested separately and then the resulting lists are combined, making a logical OR among them.

Subnet Used to select an IP subnetwork to be included in the collection definition. All IP nodes that are part of the selected subnet will be included in the collection set.

4.3 Using the Collection Editor

The Collection Editor is the /usr/OV/bin/collectioned program. It can be called from the command line or by selecting Tools→Collection Editor... from the NetView for AIX menu bar. The Collection Editor dialog box shows a list of all available collections. Figure 34 on page 55 shows the Collection Editor.



Collection: Description:

```
itso6611.itso.ral.ibm.com:te0_dependents     Collection o
itso6611.itso.ral.ibm.com:tk1_dependents     Collection o
mlmDomain_Default                            Default mlm
mlmDomain_rs600013.itso.ral.ibm.com          mlmDomain Co
mlmGroup                                     The mlm grou
siaGroup                                     The sia grou
slmGroup                                     The slm grou
thingy
```

Editor Format: [✓ Dialog] Text Editor

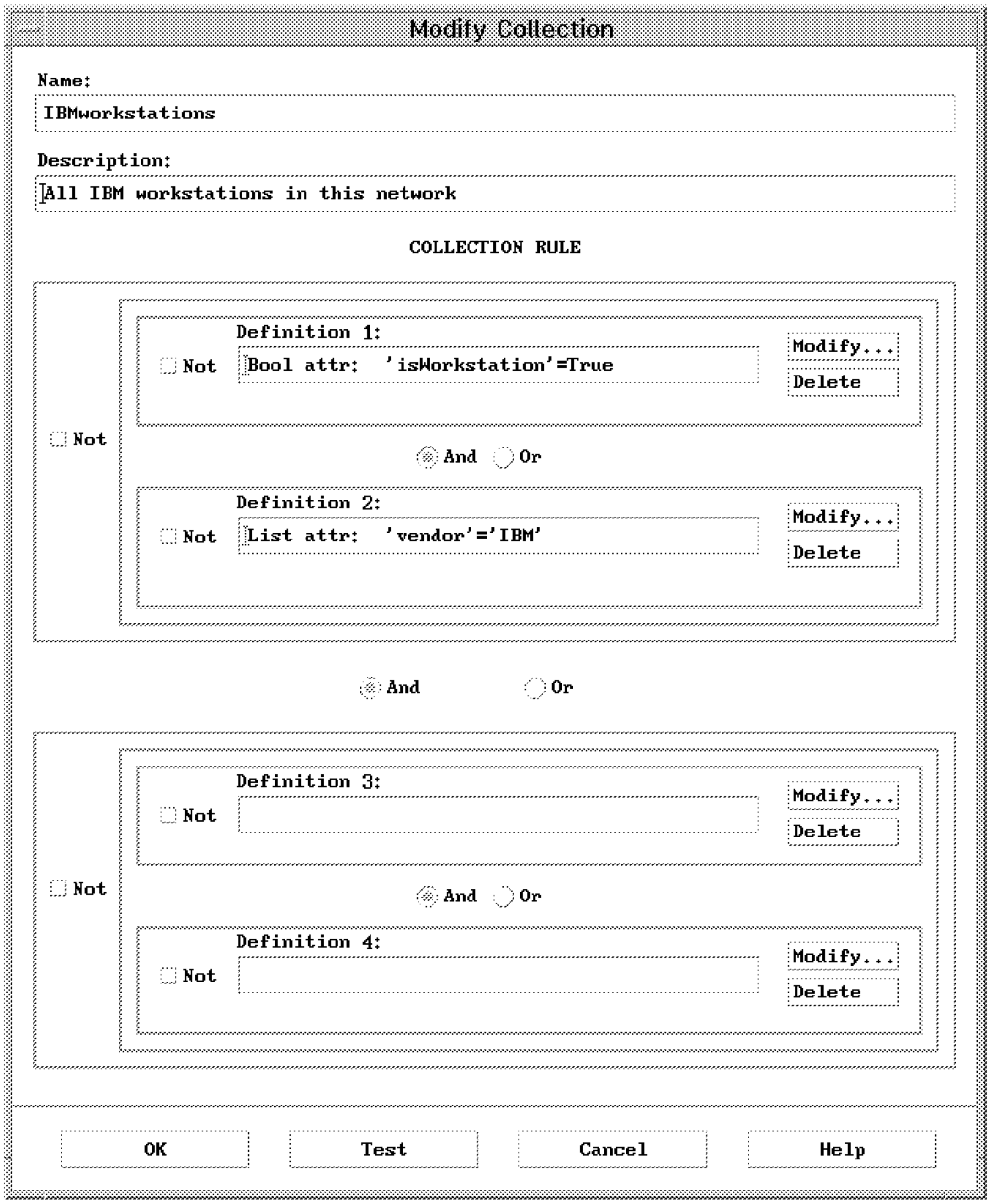
Figure 34. Collection Editor Dialog Box

The editor has five main functions that can be selected from the push buttons on the right side of the dialog box:

Add...	Used to add a new collection to the collection list.
Modify...	Modify the characteristics of the selected collection.
Copy...	Copy a selected collection to a new one.
Resolve...	Shows a list with all objects of the selected collection.
Delete	Remove the selected collection from list.

The editor can work in one of two formats, selectable using the radio buttons. These formats affect how the user accesses the information contained in the collection. In Figure 35 on page 56 there is an example showing how a collection is organized in the Dialog format. Figure 36 on page 57 shows the same collection in the Text format.

The Dialog format is useful in most cases and is easier to use than the Text format, but the Text format can compose more complex rules.


Modify Collection

Name:
IBMworkstations

Description:
All IBM workstations in this network

COLLECTION RULE

☐ Not

Definition 1:
☐ Not Bool attr: 'isWorkstation'=True

[✓ And] Or

Definition 2:
☐ Not List attr: 'vendor'='IBM'

[✓ And] Or

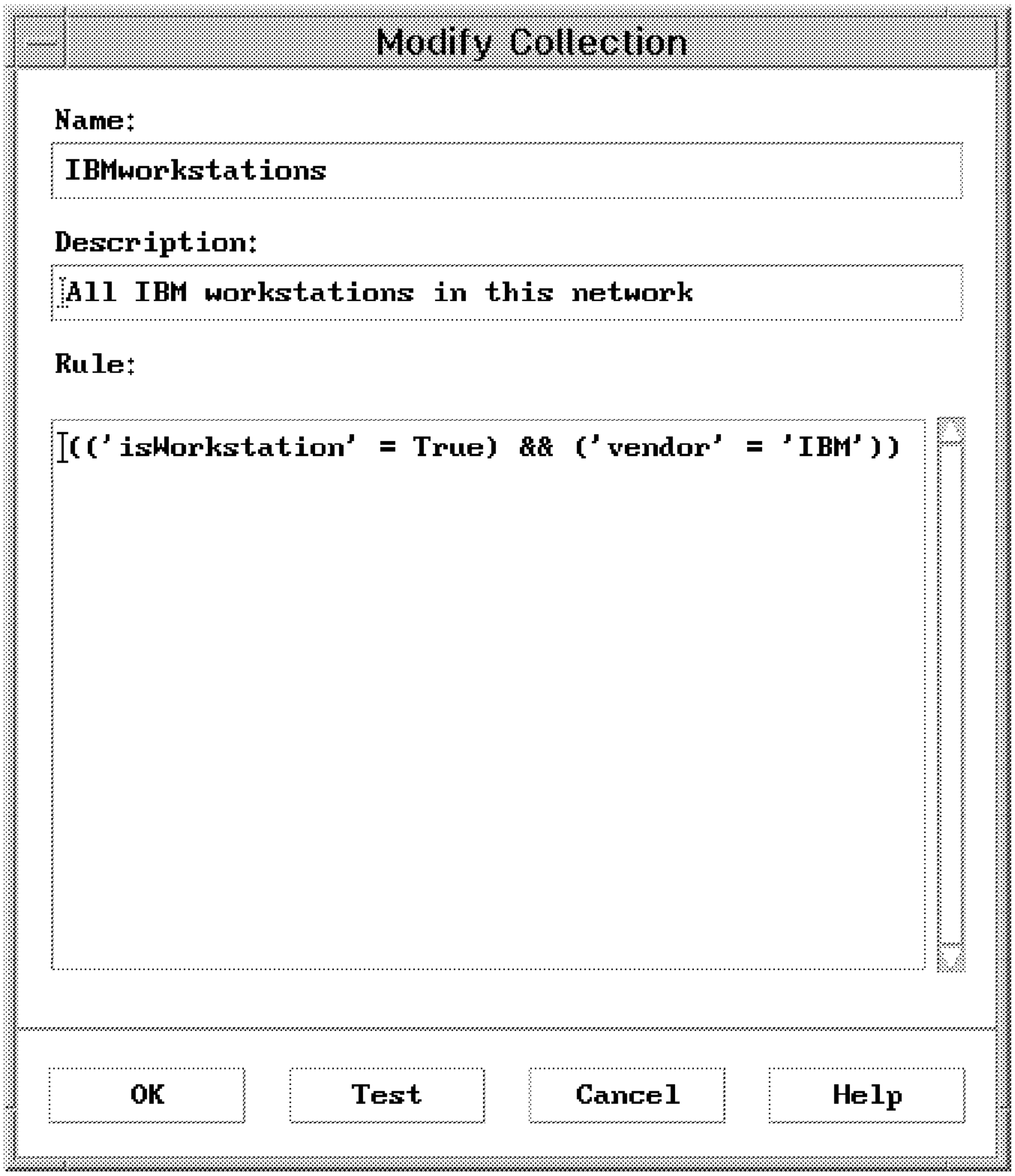
☐ Not

Definition 3:
☐ Not

[✓ And] Or

Definition 4:
☐ Not

Figure 35. Collection in Dialog Format



Name:
IBMworkstations

Description:
All IBM workstations in this network

Rule:
```
(('isWorkstation' = True) && ('vendor' = 'IBM'))
```

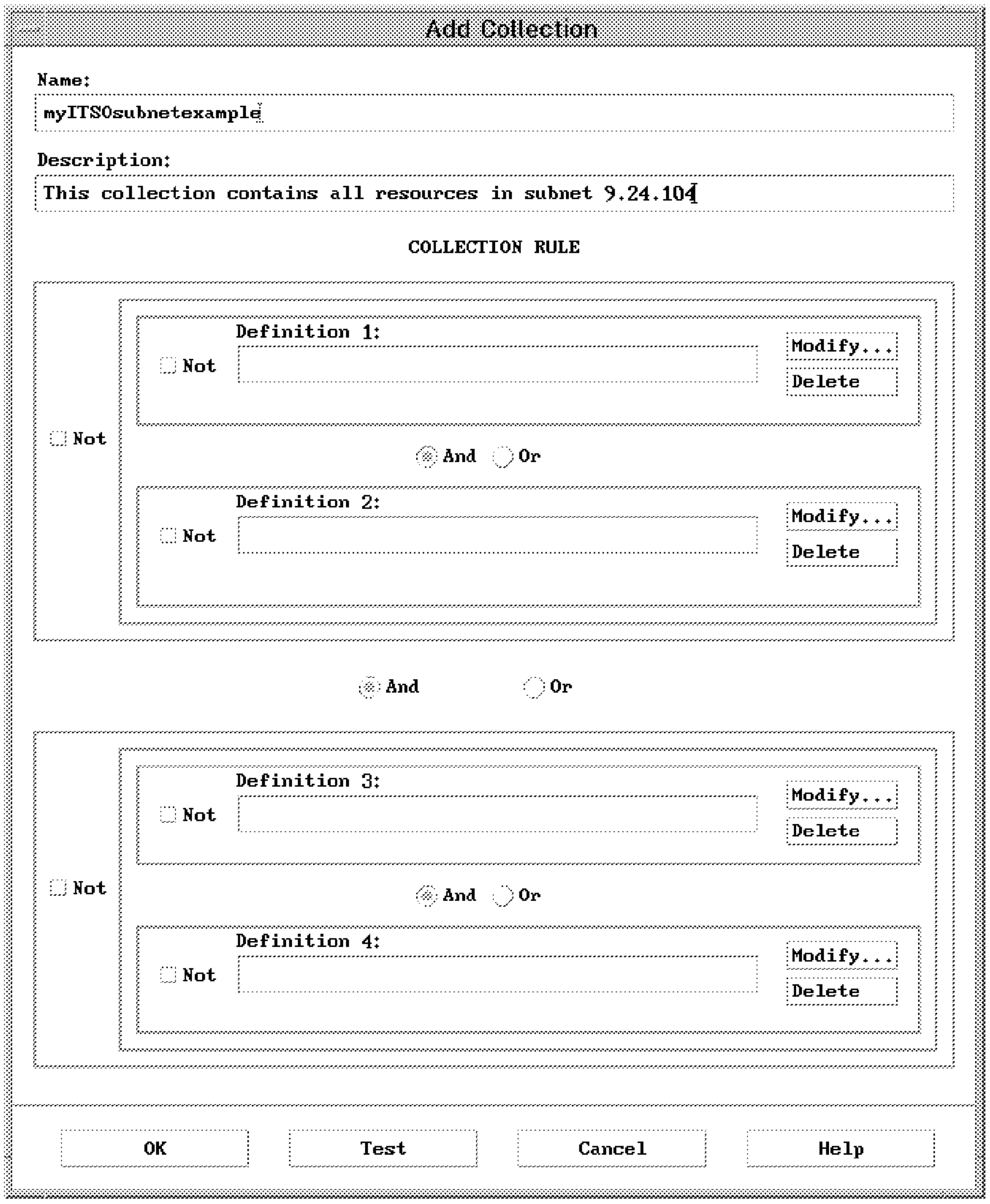
Figure 36. Collection in Text Format

4.4 Using the EUI to Create a New Collection

To open the Collection Editor using the NetView for AIX pull-down menu select Tools→Collection Editor. Click on **Add** to add a new collection to the collection list. The Add Collection dialog box can be seen on Figure 37 on page 58.

Name	Add a name for the new collection. It will be used in all references to the collection, like delete or modify. The name must be different of all other names in the collection list.
Description	Used to add meaningful comments, related to the collection resources.

Not, And and Or	Set the logical operation among the descriptions.
Modify	Add or modify the definition type, opening the Modify Definition dialog box.
Delete	Clear the definition field.
Test	Used to verify if the collection has the correct elements, according to the definitions and the logical expression.


Add Collection

Name:
myITSOsubnetexample

Description:
This collection contains all resources in subnet 9.24.104

COLLECTION RULE

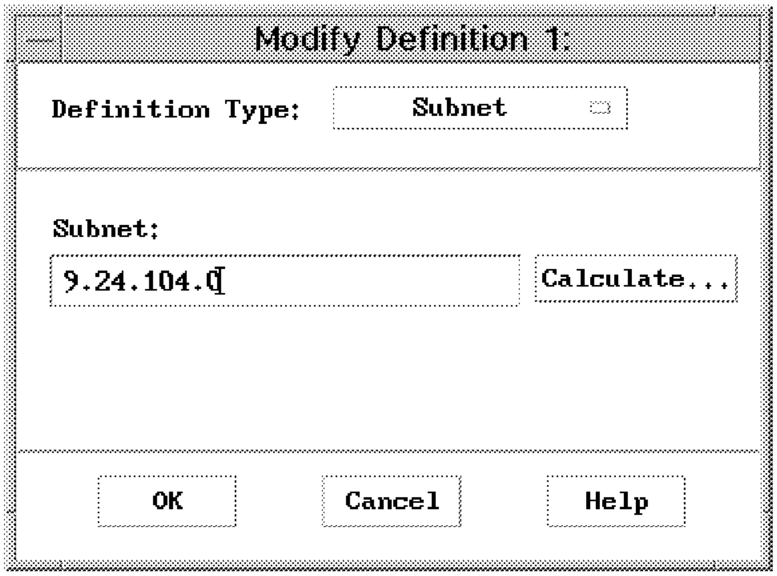
Figure 37. Adding a New Collection Using the Collection Editor

The following sections will show some examples of how to create a definition.

4.4.1 Creating a Subnet Collection

The subnet collection is a set of all resources of a particular IP subnetwork. To create it, follow these steps:

1. Open the Collection Editor and select **Add**.
2. Enter the name and description of the new collection.
3. Select **Modify** in the Definition 1 field. The Modify Definition dialog box will appear.
4. Choose **Subnet** in the Definition Type menu. Figure 38 shows the Modify Definition dialog box after selecting the subnet option.

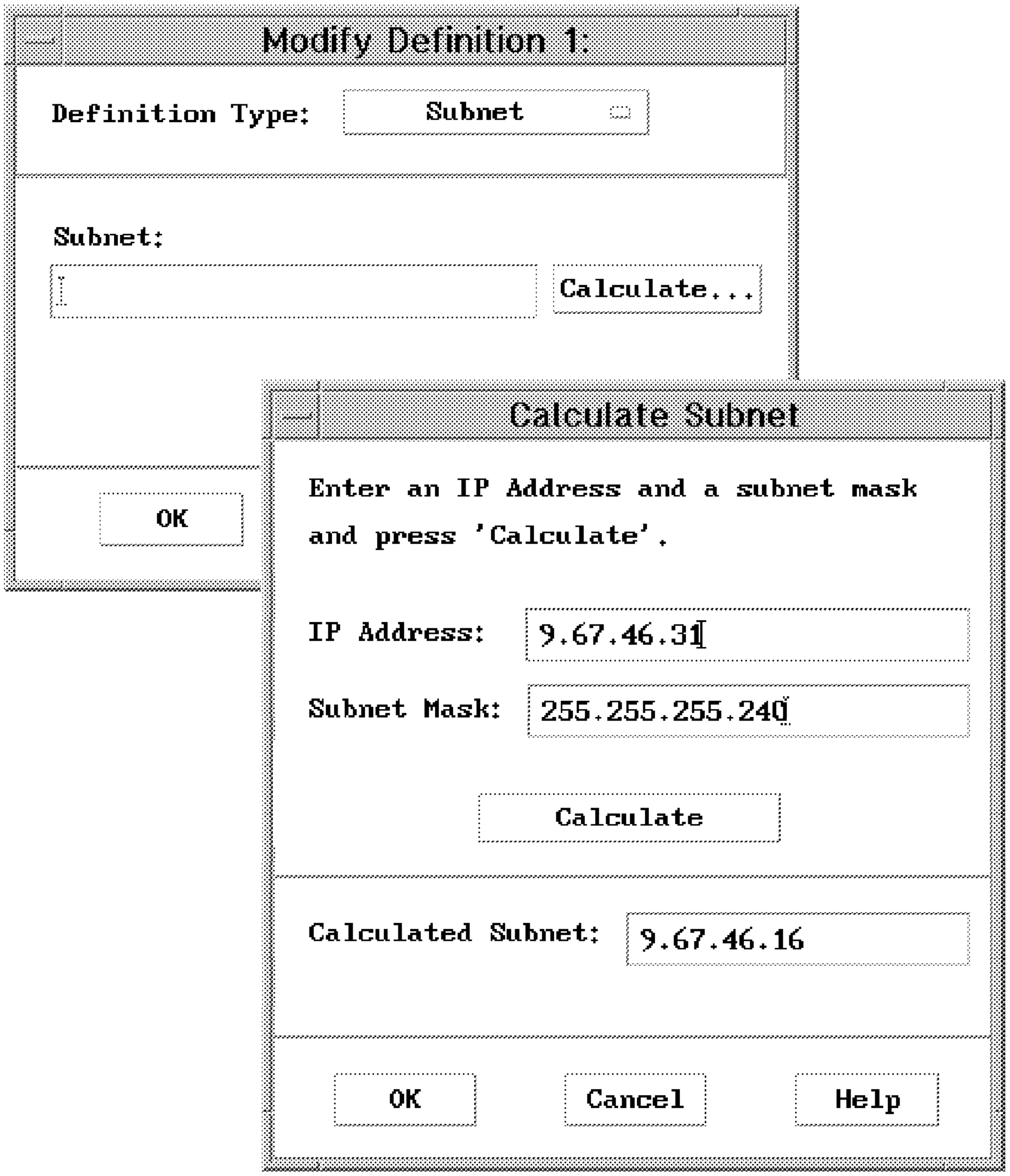

Modify Definition 1:

Definition Type: Subnet

Subnet:
9.24.104.0

Figure 38. Creating the Subnet Collection

5. Notice that the subnet address that you enter must include all four elements (9.24.104.0 in our class C subnet example). If you are unsure of what to enter, you may wish to click on **Calculate** to help you. It calculates the address based on an IP address and a mask. Figure 39 on page 60 shows an example of how to use it.



Definition Type: Subnet

Subnet:

Enter an IP Address and a subnet mask and press 'Calculate'.

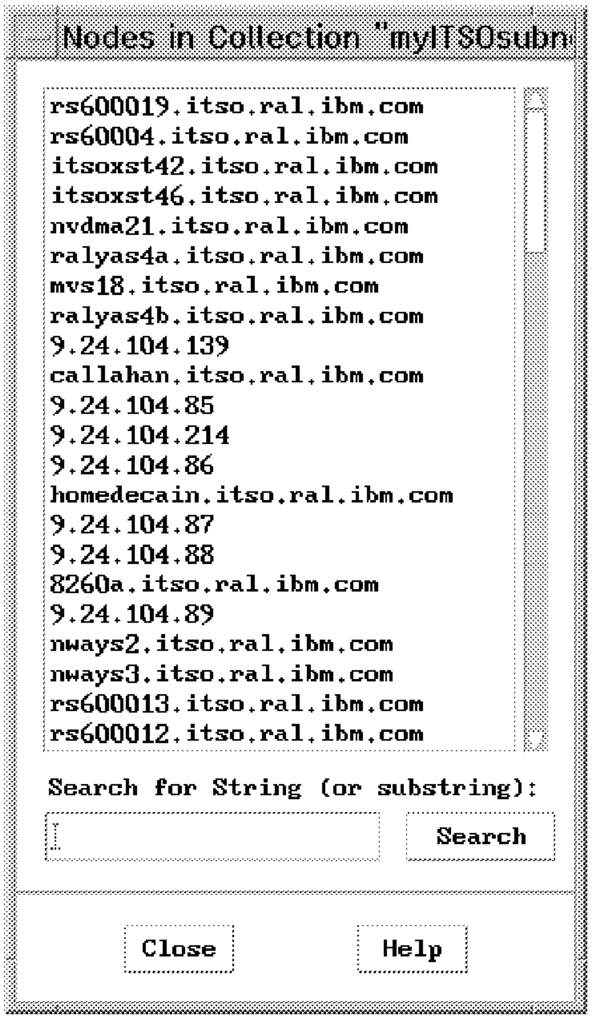
IP Address: 9.67.46.31

Subnet Mask: 255.255.255.240

Calculated Subnet: 9.67.46.16

Figure 39. Calculating a Subnet Based on an IP Address

The Test button can be used to see all resources that match the collection definition before its creation. It is very useful to verify each step when creating a complex definition. Figure 40 on page 61 shows the result of this collection after selecting Test.



```
rs600019.itso.ral.ibm.com
rs60004.itso.ral.ibm.com
itsoxst42.itso.ral.ibm.com
itsoxst46.itso.ral.ibm.com
nvdma21.itso.ral.ibm.com
ralyas4a.itso.ral.ibm.com
mvs18.itso.ral.ibm.com
ralyas4b.itso.ral.ibm.com
9.24.104.139
callahan.itso.ral.ibm.com
9.24.104.85
9.24.104.214
9.24.104.86
homedecain.itso.ral.ibm.com
9.24.104.87
9.24.104.88
8260a.itso.ral.ibm.com
9.24.104.89
nways2.itso.ral.ibm.com
nways3.itso.ral.ibm.com
rs600013.itso.ral.ibm.com
rs600012.itso.ral.ibm.com
```

Figure 40. Result Obtained for the Subnet Using Test Button

6. Select **OK** to add the new collection to the collection list.

4.4.2 Creating a Node List Collection

This option is an easy way to create a collection based on the objects selected on the map. If desired, an object can be directly inserted into the list, by clicking on the **Add** button.

To put the selected objects into the collection List of Nodes, do the following:

1. Select all desired objects from the NetView for AIX submaps, as show in Figure 41 on page 62 (you can select multiple nodes by holding down the Ctrl key while you select them with the left mouse button).
2. Start the Collection Editor. .
3. Select **Add** to insert a new collection in the collection list.
4. Enter the name and description of the new collection.
5. Click on **Modify** to pop-up the Modify Definition dialog box.
6. Select **Node List** in the Definition Type menu. The dialog box changes to the format showed in Figure 42 on page 63.

7. Select **Add From Map** to include all selected objects from all submaps into the List of Nodes.

8. To remove a node from the list, select it from the list and click on **Delete**.

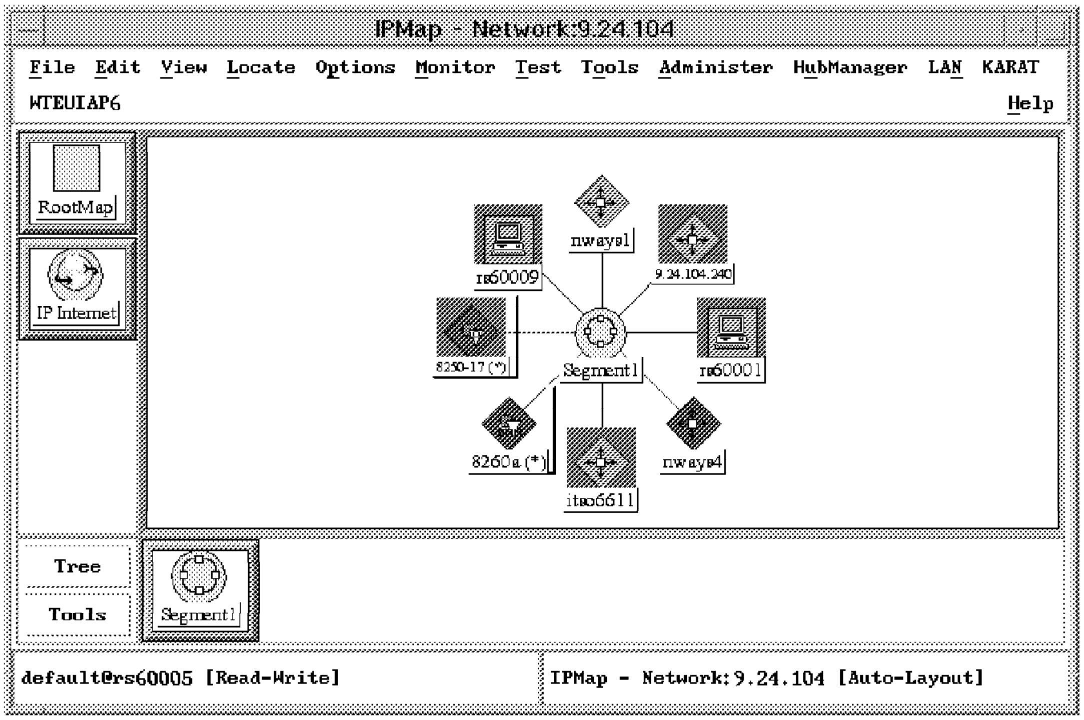
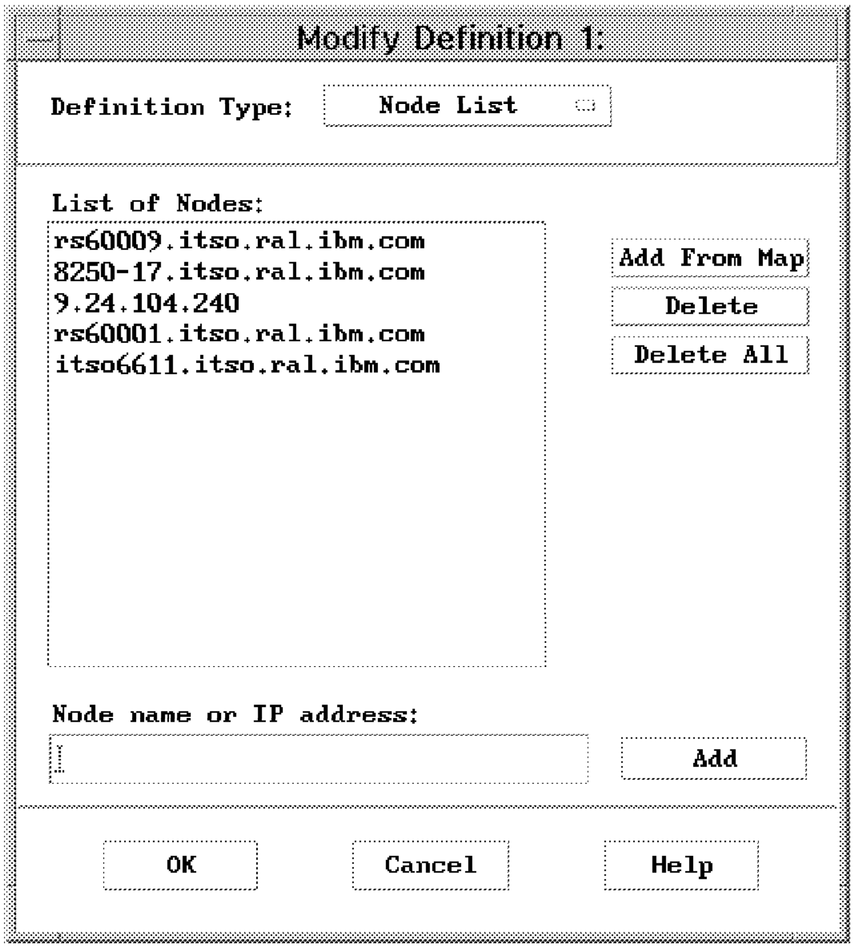


Figure 41. Objects Selected on Map



```
rs60009.itso.ral.ibm.com
8250-17.itso.ral.ibm.com
9.24.104.240
rs60001.itso.ral.ibm.com
itso6611.itso.ral.ibm.com
```

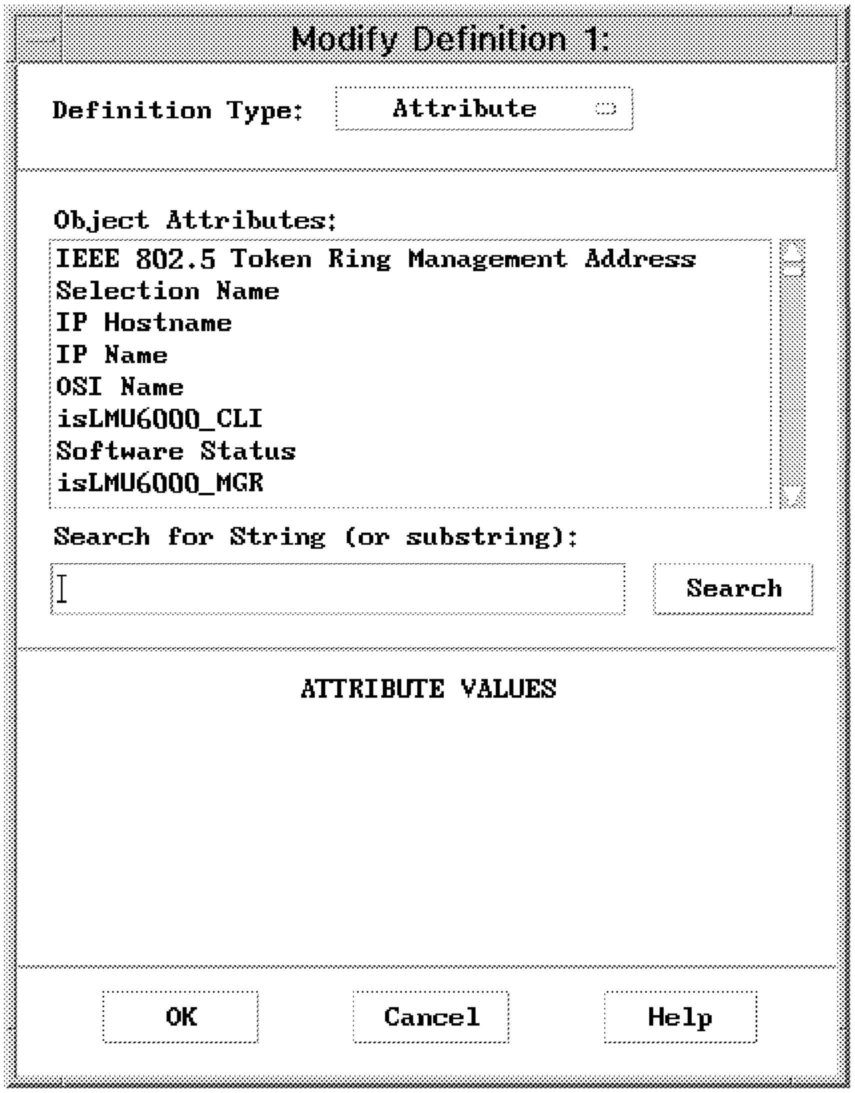
Figure 42. List of Objects in the Modify Definition Dialog Box

4.4.3 Creating an Attribute Collection

The Attribute collection option has a wide scope of definitions and can be used to create a more complex expression to select specific resources. Any fields in the NetView for AIX object database may be used to define the collection rule. This includes not just the standard fields that are provided by NetView for AIX, but also any fields of your own that you may wish to define.

We will illustrate the way that Attribute-based collections work by means of an example. In this case we will create a collection containing all hubs and all workstations. To create this collection, do the following:

1. Start the Collection Editor by selecting Tools→Collection Editor from the menu bar.
2. Click on **Add** to begin editing a new collection.
3. Enter the name and the description of the new collection.
4. Select **Modify** in the Definition 1 field. The Modify dialog box will pop up.
5. Change the Definition Type to **Attribute**. Figure 43 on page 64 shows the Modify Definition dialog box format.


Modify Definition 1:

Definition Type: Attribute

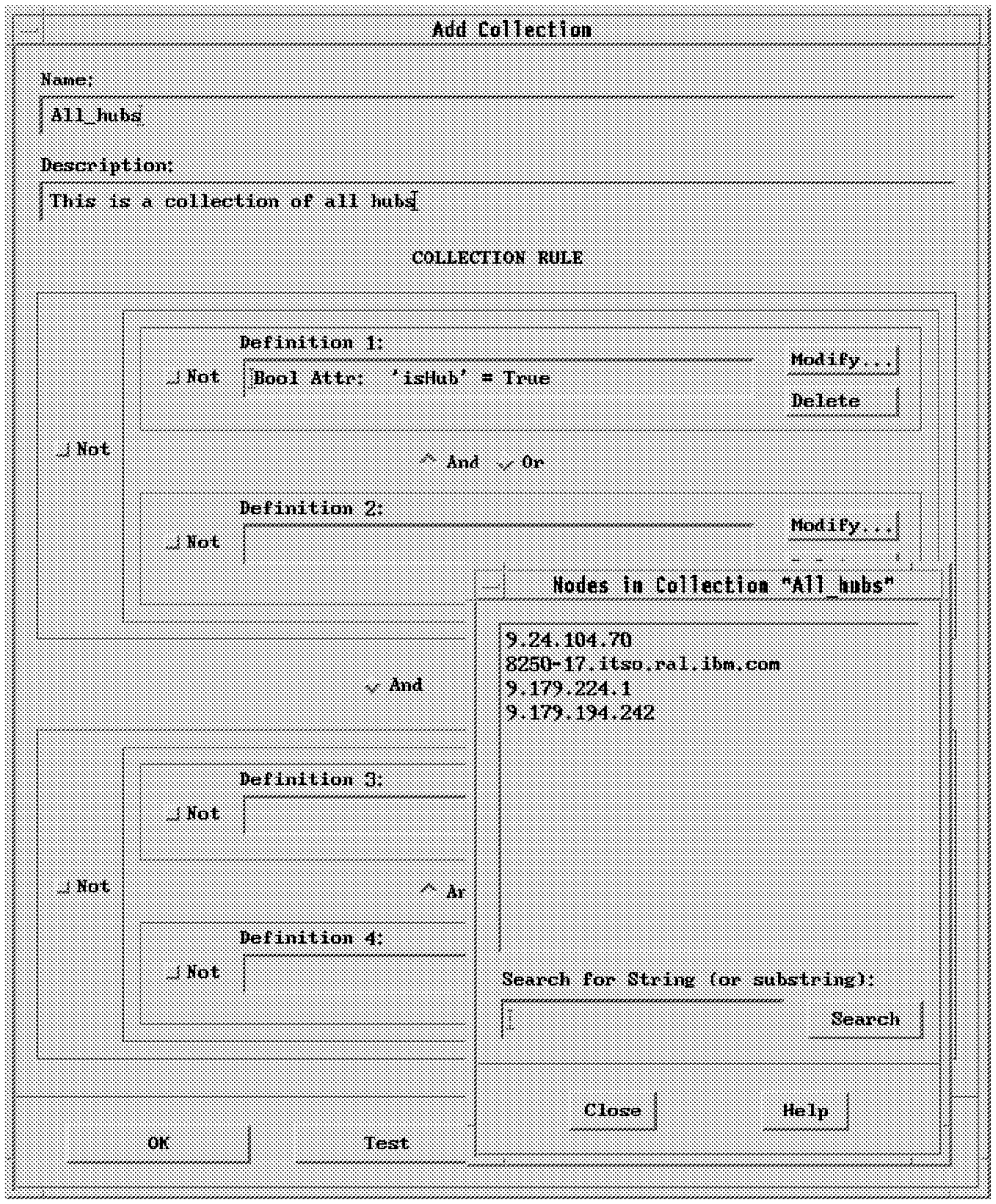
Object Attributes:

IEEE 802.5 Token Ring Management Address
Selection Name
IP Hostname
IP Name
OSI Name
isLMU6000_CLI
Software Status
isLMU6000_MGR

Search for String (or substring):

Figure 43. Attribute Format for Modify Definition Dialog Box

6. Select isHub in the Object Attributes list.
7. Select **True** for the Boolean value.
8. Click on **OK**.
9. Select **Test** to see all hubs of this network. Figure 44 on page 65 shows the result we obtained.



Add Collection

Name:

All_hubs

Description:

This is a collection of all hubs

COLLECTION RULE

Definition 1: Not Bool Attr: 'isHub' = True

Nodes in Collection "All_hubs"

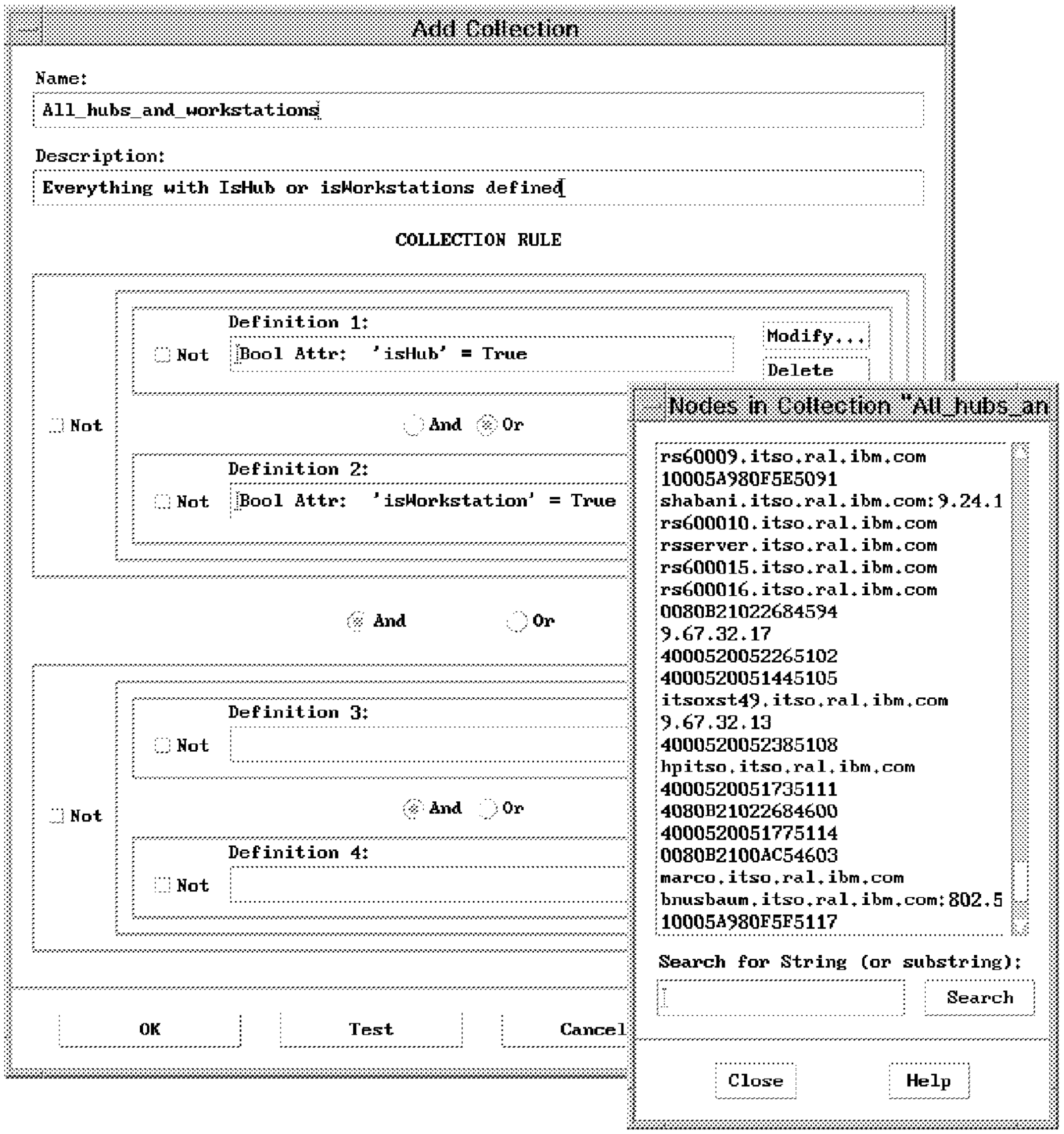
```
9.24.104.70
8250-17.itso.ral.ibm.com
9.179.224.1
9.179.194.242
```

Figure 44. Collection of All Hubs

10. Click on **Close** to close the list of nodes.
11. Click on the **Modify** push button in the Definition 2 field. The Modify Definition dialog box will pop-up.
12. Select isWorkstation in the Object Attributes list.
13. Select **True** for the Boolean value.
14. Click on **OK**.
15. Select the **Or** radio button between Definition 1 and Definition 2.
16. Select **Test** to see the result. Figure 45 on page 66 shows the resulting list of nodes.

17. Click on **Close**.

18. Select **OK** to create the collection and add it into the collections list.



Add Collection

Name:
All_hubs_and_workstations

Description:
Everything with IsHub or isWorkstations defined

COLLECTION RULE

[ ] Not
Definition 1:
[ ] Not Bool Attr: 'isHub' = True
Modify... Delete
( ) And (● Or)
Definition 2:
[ ] Not Bool Attr: 'isWorkstation' = True

(● And) ( ) Or

[ ] Not
Definition 3:
[ ] Not
(● And) ( ) Or
Definition 4:
[ ] Not

OK Test Cancel

Nodes in Collection "All_hubs_an

```
rs60009.itso.ral.ibm.com
10005A980F5E5091
shabani.itso.ral.ibm.com:9.24.1
rs600010.itso.ral.ibm.com
rsserver.itso.ral.ibm.com
rs600015.itso.ral.ibm.com
rs600016.itso.ral.ibm.com
0080B21022684594
9.67.32.17
4000520052265102
4000520051445105
itsoxst49.itso.ral.ibm.com
9.67.32.13
4000520052385108
hpitso.itso.ral.ibm.com
4000520051735111
4080B21022684600
4000520051775114
0080B2100AC54603
marco.itso.ral.ibm.com
bnusbaum.itso.ral.ibm.com:802.5
10005A980F5F5117
```

Search for String (or substring):

Search

Close Help

Figure 45. Collection of All Hubs and Workstations

We can see the same list in a graphical way by going to the Collections submap (under the Root submap) and selecting our new collection. The result is shown in Figure 46 on page 67.

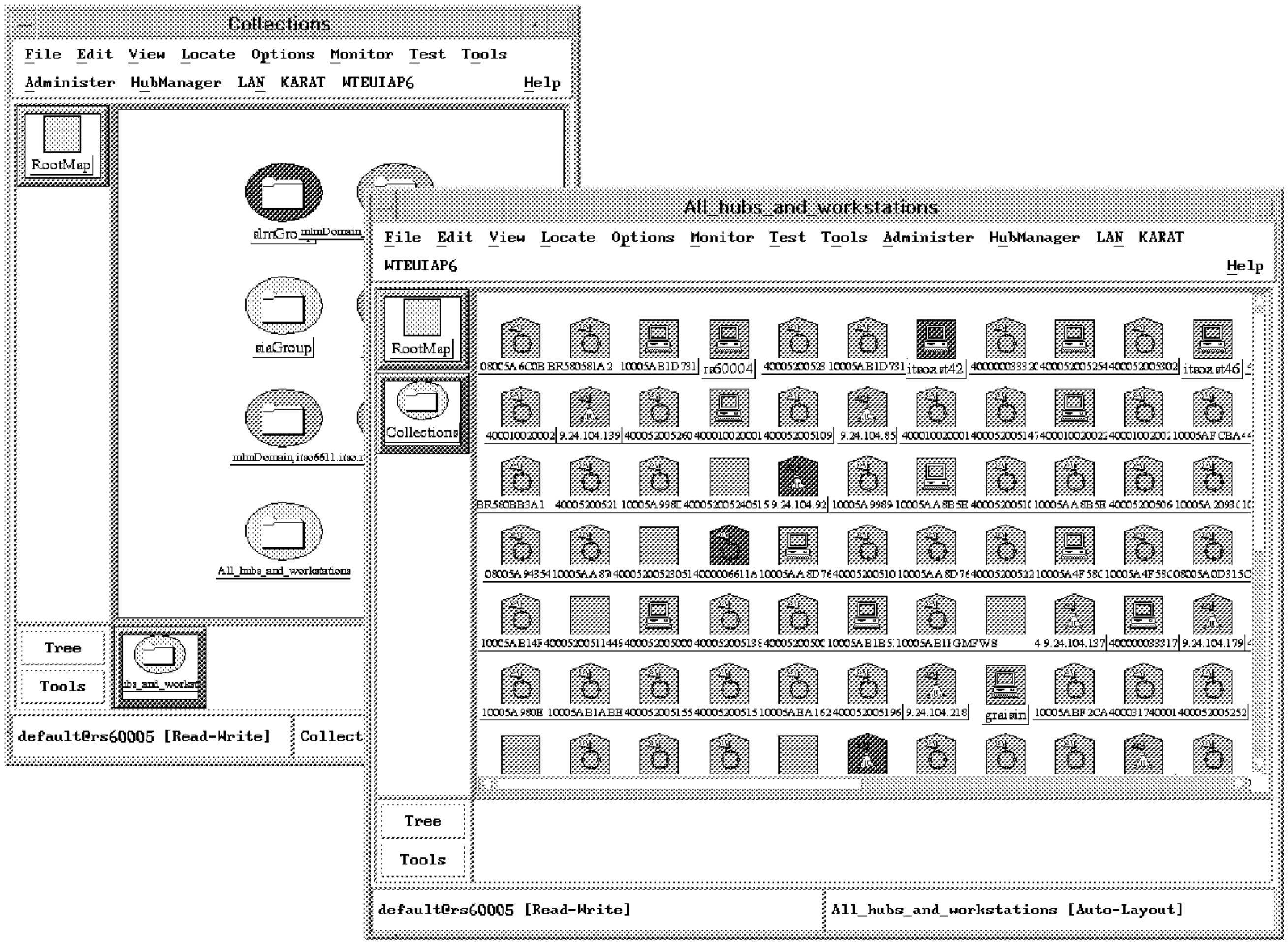

Collections
File Edit View Locate Options Monitor Test Tools
Administer HubManager LAN KARAT WTEUIAP6 Help
RootMap
Tree
Tools
default@rs60005 [Read-Write]
All_hubs_and_workstations
File Edit View Locate Options Monitor Test Tools Administer HubManager LAN KARAT
WTEUIAP6 Help
RootMap
Collections
Tree
Tools
default@rs60005 [Read-Write]
All_hubs_and_workstations [Auto-Layout]

Figure 46. Collection of All Hubs and Workstations Submap. Notice that the status colors are automatically propagated to this map.

4.5 The Collection APIs

The API calls allow programs to add, delete, modify and verify contents of collections in the collections list. A notification method is also available for programs to register for changes to a collection.

The Collection Facility is composed of a new daemon and library. The daemon manages collections and rules and is called nvcold. The library contains the object code of the Collection Facility API and is called libcollection.a. The headers files that must be included on all C programs that use the Collection APIs are:

/usr/OV/include/OV/nvCollection.h

/usr/OV/include/OV/nvCollectionErrs.h

Table 3 on page 68 summarizes the functions available and the API calls that implement those functions. All APIs are fully documented in *NetView for AIX*

Programmer's Reference, SC31-8165 (also provided in Dynatext for online browsing).

Table 3. Collection Facility API Summary

Function Type	**API Calls**
Open and close a connection to the nvcold daemon	nvCollectionOpen nvCollectionXOpen nvCollectionDone nvCollectionXDone
Add, Modify or Delete a Collection Rule	nvCollectionAdd nvCollectionModify nvCollectionDelete
Obtain a list of all resources in a collection	nvCollectionResolve nvCollectionEvaluate
OR two or more collections together	nvCollectionUnion nvCollectionListUnion
AND two or more collections together	nvCollectionIntersect nvCollectionListIntersect
Get information about defined collections	nvCollectionGetInfo nvCollectionListCollections nvCollectionGetTimestamp
List all collections that a resouce is in	nvCollectionGetAllForObject
Define a routine to be called when a collection changes	nvCollectionAddCallback nvCollectionRead
Handle API Errors	nvCollectionError nvCollectionErrorMsg
Memory Management	nvCollectionFreeDefn nvCollectionFreeList nvCollectionFreeChangeList

4.5.1 The Grammar of the Collection Rules

Programs using the APIs can work with collections previously defined using the EUI ruleset editor. However, if you want to create new collection rules using the API you have to provide the rule definition in text form. Collection rules defined in this way must adhere to a defined grammar. The grammar is as follows:

```
IN            <list of objects>
IN_COLLECTION <name of collection>
IN_SUBNET     <subnet>
<field> <op> <value>
```

- The IN rule can be used to define a very specific collection where the collection will be made up of the objects whose selection names are specified in the list. If a list is provided, it needs to be enclosed in single quotes.
- The IN_COLLECTION rule can be used to define new collections based on the rule of an already existing collection.
- The IN_SUBNET rule can be used to define a collection of objects based on the subnet in which the objects exist.
- The <field> <op> <value> rule can be used to define a rule based on the fields in the object database. <op> can be one of the following: =, !=,

<, >, <=, >=. If the <field> is composed of more than one string (for example, SNMP sysObjectID), then it needs to be enclosed in single quotes.

In addition, the rules can be made more complex by joining simple rules with AND, OR, !, and parenthesis. The actual keywords that can be used include AND, &&, OR, ||, !, (,).

4.6 Practical Examples

We will now show several examples of programs that use the collections API. At the end of the section (4.7, “The wtcoll Sample Program” on page 76) we introduce the wtcoll sample program which brings together all of the individual examples into one utility program.

4.6.1 Makefile

Figure 47 shows the makefile used to compile the wtcoll.c program, linking the libraries libovw.a, libov.a and libcollection.a. All other C examples in this chapter use a makefile exactly like this one, changing only the C program name.

```
INCDIRS =
LDRDIRS =
LIBS = -lovw -lov -lcollection

CC = /bin/cc

CFLAGS = -g $(INCDIRS) $(LDRDIRS)

SRCFILES = wtcoll.o

all: wtcoll

wtcoll: $(SRCFILES)
        $(CC) $(CFLAGS) $(SRCFILES) -o wtcoll $(LIBS)
```

Figure 47. Makefile for the wtcoll Sample

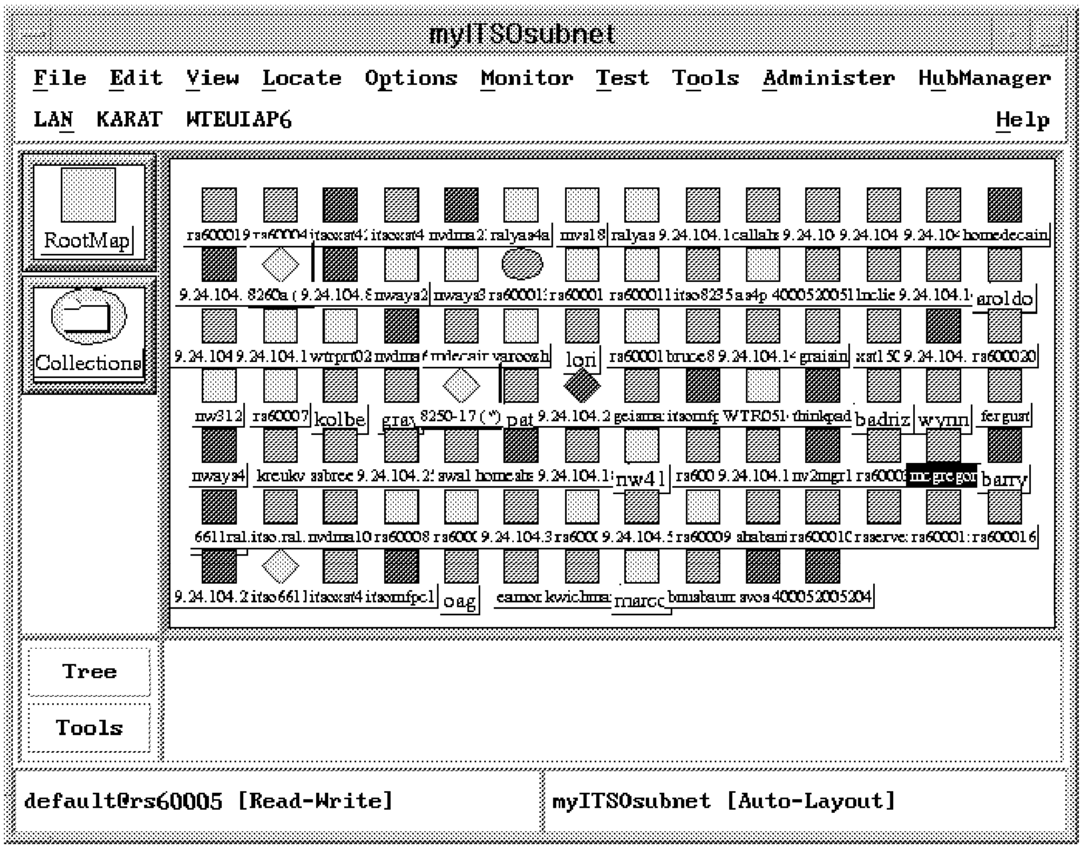
4.6.2 Using the APIs to Create a Subnet Collection

The simple example in Figure 48 on page 70 shows how to create a subnet collection using the API. It first opens the connection to nvcold, creates and adds the collection rule, and then handles any error that may occur.

```
#include <stdio.h>
#include "/usr/OV/include/OV/ovw_obj.h"
#include "/usr/OV/include/OV/nvCollection.h"
#include "/usr/OV/include/OV/nvCollectionErrs.h"

void main()
{
  /*
     collectionName - used to refer to the new collection
     collectionDescription - used to describe/comment the new collection
     subnetName     - used to define which subnet is the aim of the
                      collection
     collectionRule - holds the rule used to create the collection
  */

  static char *collectionName = "myITSOsubnet";
  static char *collectionDescription = "subnet on ITSO";
  static char *subnetName = "9.24.104.0";
  char collectionRule [100];

  int  collectionFD;           /* collection file descriptor */
  int  collectionConnectionFD; /* collection connection descriptor */

  /* Estabilishes a connection with the Collection Facility daemon */
  collectionConnectionFD = nvCollectionOpen();

  /* Creates the collection rule */
  strcpy( collectionRule, "IN_SUBNET " );
  strcat( collectionRule, subnetName );

  /* Add the new collection to the collections list */
  /* It may take some CPU processing if the collection is large */
  collectionFD = nvCollectionAdd( collectionName,
                                  collectionDescription,
                                  collectionRule,
                                  1 );

  /* Verifies if the collection has been successfully created */
  if ( collectionFD != NV_COLLECTION_SUCCESS )
    fprintf( stderr, "Unsucessfull operation: %s\n",
             nvCollectionErrorMsg(nvCollectionError()));
  else
    fprintf( stderr, "Collection %s created!\n", subnetName );

  nvCollectionDone();
}
```

Figure 48. Example of Adding a Collection Rule Using the API

Figure 49 on page 71 shows the resulting submap under the Collections icon for the collection myITSOsubnet created by this program.


myITSOsubnet
File Edit View Locate Options Monitor Test Tools Administer HubManager
LAN KARAT WTEUIAP6 Help
RootMap
Collections
Tree
Tools

default@rs60005 [Read-Write]

myITSOsubnet [Auto-Layout]

Figure 49. Nodes in Collection myITSOsubnet

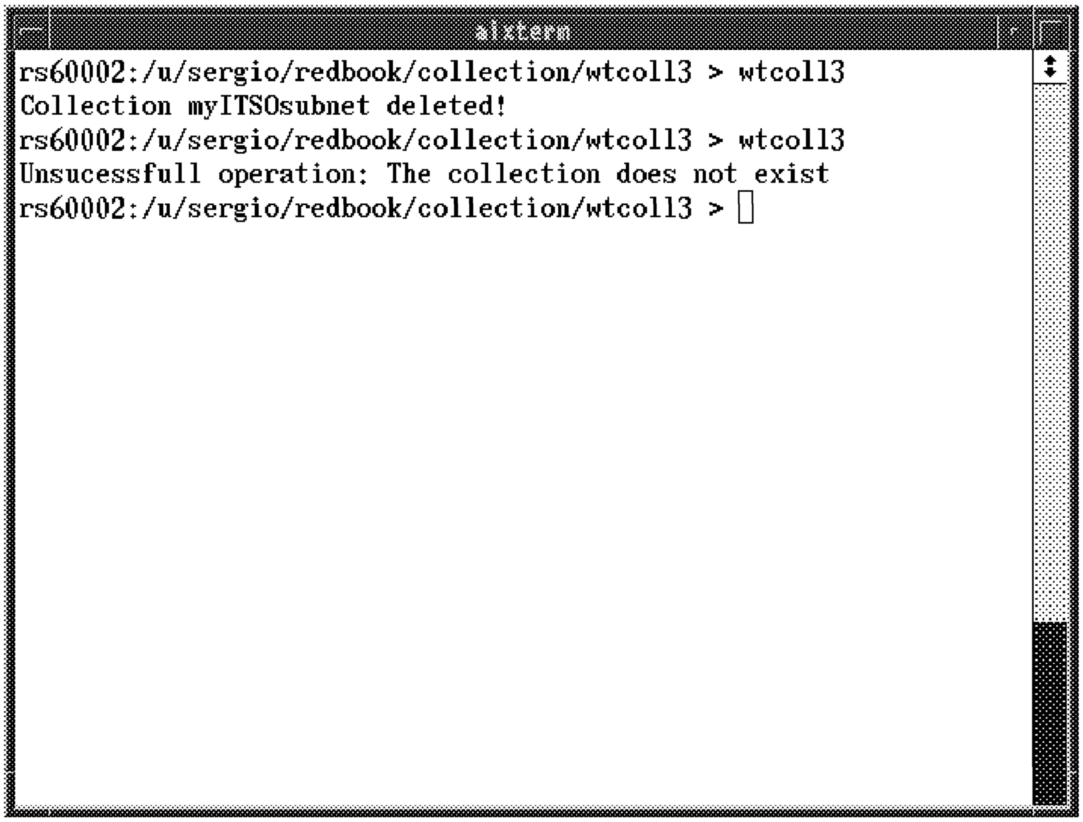
4.6.3 Using the APIs to Delete a Collection

In Figure 50 on page 72, we use the nvCollectionDelete call to show how to remove an existing collection.

```
#include <stdio.h>
#include "/usr/OV/include/OV/ovw_obj.h"
#include "/usr/OV/include/OV/nvCollection.h"
#include "/usr/OV/include/OV/nvCollectionErrs.h"

void main()
{
  /*
     collectionName - used to refer to the collection
  */

  static char *collectionName = "myITSOsubnet";

  int  collectionFD;           /* collection file descriptor */
  int  collectionConnectionFD; /* collection connection descriptor */

  /* Estabilishes a connection with the Collecion Facility daemon */
  collectionConnectionFD = nvCollectionOpen();

  /* Add the new collection to the collections list */
  /* It may take some CPU processing if the collection is large */
  collectionFD = nvCollectionDelete( collectionName , 1 );

  /* Verifies if the collection has been sucessfully created */
  if ( collectionFD != NV_COLLECTION_SUCCESS )
    fprintf( stderr, "Unsucessfull operation: %s\n",
      nvCollectionErrorMsg(nvCollectionError()));
  else
    fprintf( stderr, "Collection %s deleted!\n", collectionName );

  nvCollectionDone();
}
```

Figure 50. Example of Removing a Collection Using the API

Figure 51 on page 73 shows the result of running this program twice: once to delete the collection we added in the previous example and then again, thereby generating an error message.



```
rs60002:/u/sergio/redbook/collection/wtcoll3 > wtcoll3
Collection myITSOsubnet deleted!
rs60002:/u/sergio/redbook/collection/wtcoll3 > wtcoll3
Unsucessfull operation: The collection does not exist
rs60002:/u/sergio/redbook/collection/wtcoll3 > 
```

Figure 51. Result of Deleting a Collection

4.6.4 Using the APIs to Execute a Command on All Objects in a Collection

An important Collection Facility feature is its capability to execute operations over each object. In this example we will show how to send a trap to each object, using a script file and a C program. We have left some functions to be executed in the script file to gain more flexibility by allowing us to re-use the C program in other examples.

First, the C program extracts all object IDs from the collection using nvCollectionResolve. This is shown in Figure 52 on page 74.

```
#include <stdio.h>
#include <OV/ovw.h>
#include <ctype.h>

#include "/usr/OV/include/OV/ovw_obj.h"
#include "/usr/OV/include/OV/nvCollection.h"
#include "/usr/OV/include/OV/nvCollectionErrs.h"

#define  EVALUATE  0
#define  RESOLVE   1

   /*
      Function Prototypes
   */

OVwObjectIdList *getObjectIdList( char *collectionName ,
                                  int  eval_resol );
void printObjectIds( char *collectionName , int resolution );


/*
   getObjectIdList  -  get a list of objects of a collection
       char *collectionName is the name of the collection
       int  eval_resol - EVALUATE => evaluate the collection again
                         RESOLVE  => only resolve the collection
*/
OVwObjectIdList *getObjectIdList( char *collectionName ,
                                  int  eval_resol )
{
  int             collectionConnectionFD; /* collection descriptor */
  char            *description;
  char            *rule;
  OVwObjectIdList *objectList;

  /* Estabilishes a connection with Collecion Facility daemon */
  collectionConnectionFD = nvCollectionOpen();

  if ( eval_resol == RESOLVE )
            /* resolve for collection name */
    objectList = nvCollectionResolve( collectionName );
  else
    {
                /* find collection rule */
      nvCollectionGetInfo( collectionName, &description, &rule );
                /* and evaluates the collection */
      objectList = nvCollectionEvaluate( rule );
    }

  free(description);   /* Free memory */
  free(rule);          /* Free memory */

  nvCollectionDone();  /* Close connection with the
                          Collection Facility daemon */

  return objectList;
}


/*
   printObjectIds  -  print the id of all objects
                      in a collection
*/
void printObjectIds( char *collectionName , int resolution )
{
  int             count = 0;
  OVwObjectIdList *objList;

  /* Get the object list */
```

Figure 52 (Part 1 of 2). Example of Listing the Resources in a Collection Using the API

```
  objList = getObjectIdList( collectionName, resolution );
  /* print all the objectIds */
  for ( count = 0 ; count < objList->count ; count++ )
    fprintf( stdout, "%d\n", objList->object_ids[count] );

  /* Free the object list */
  OVwDbFreeObjectIdList( objList );
}


/*
   main function
*/
void main( int argc , char **argv )
{
  char     collectionName[512];

  OVwDbInit();

  strcpy( collectionName, "myITSOsubnet" );

  printObjectIds( collectionName, RESOLVE );
}
```

Figure 52 (Part 2 of 2). Example of Listing the Resources in a Collection Using the API

The program in Figure 52 on page 74 was called wtcoll4. We invoked it from the shell script shown in Figure 53.

```
#!/bin/ksh

# For all objects in wtcoll4, do
for i in `wtcoll4`
do
#  Get the Selection Name using the Object Id
#  Select only the host name, without the "." extensions
#  For example, rs60002.itso.ral.ibm.com will be changed to rs60002
   export host=$(ovobjprint -o $i |grep Selection | \
               awk '{gsub ( /"/, "" ); print $4}' | \
               awk '{gsub ( /\./, " " ); print $1}')

#  Shows the host name
   echo Sending trap to $host

#  Execute the command: send a trap
   /usr/OV/bin/snmptrap $host .1.3.6.1.4.1.2.6.3.1 $host 6 58916864 0\
   1 integer 14 2 octetstring $host 3 octetstring "Command Received"
done
```

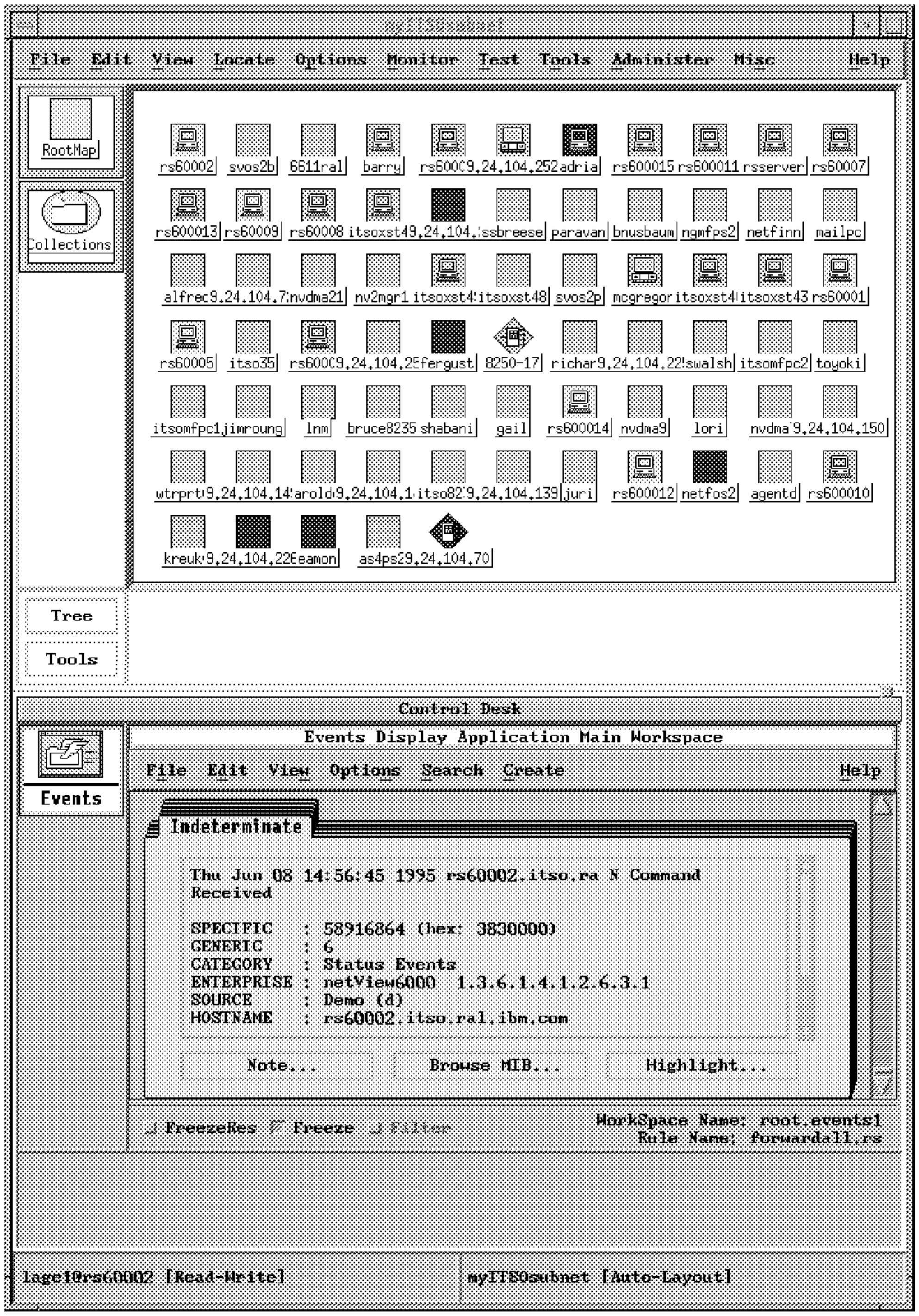
Figure 53. Shell Script that Invokes the wtcoll4 Program

The script is a simple loop that executes two commands for each node in the list returned by the wtcoll4 program.

1. It uses the ovobjprint command to extract the Selection Name attribute from the database. ovobjprint extracts all the field information from the database. The awk command is then used to extract the desired string. This step could be done in the C program, but using the object ID and the ovobjprint command inside the script gives us more flexibility to choose any object attribute we want.

2. The selection name is then used to send a trap, using the snmptrap command.

Figure 54 on page 76 is the submap with all objects of myITSOsubnet and the event received in the events application.



```
Thu Jun 08 14:56:45 1995 rs60002.itso.ra N Command
Received

SPECIFIC   : 58916864 (hex: 3830000)
GENERIC    : 6
CATEGORY   : Status Events
ENTERPRISE : netView6000  1.3.6.1.4.1.2.6.3.1
SOURCE     : Demo (d)
HOSTNAME   : rs60002.itso.ral.ibm.com
```

```
WorkSpace Name: root.events1
     Rule Name: forwardall.rs
```

```
lage1@rs60002 [Read-Write]          myITSOsubnet [Auto-Layout]
```

Figure 54. Nodes in Collection myITSOsubnet and Trap Received in rs60002

4.7 The wtcoll Sample Program

We brought together several of the collection facility API functions into a program called *wtcoll*. wtcoll is a useful utility that allows you to perform a number of operations on NetView for AIX collections from a command line.

The functions available from wtcoll are as follows:

- List all objects in a collection.

  You specify the collection name and wtcoll will resolve the collection and print a list of the objects in it.

- Create an object database field for all objects in a collection.

  The program will create a string field in the object database and place a value you specify into it for all the nodes in the collection.

- Update an object database field for all objects in a collection.
- Delete a collection.

  You specify the collection rule name and wtcoll will remove it.

- Check whether a node is a member of a given collection.

  You tell wtcoll the node selection name and the name of the collection. We show several uses for this function in Chapter 7, “Using the Collection Facility with Event Rulesets” on page 171.

You can find a listing of the the source code for wtcoll.c in Appendix B, “C Code for the wtcoll Sample Program” on page 259. Instructions on how to get a copy of the source code by anonymous FTP are printed in Appendix A, “How to Obtain the Samples in this Book” on page 257.

Chapter 5. Introducing NetView for AIX Event Rulesets

When people enter the world of systems management the first question they usually want answered is "what is happening out there?" They want to know about any significant event that may affect them, as it happens. Almost every system environment provides some kind of mechanism for this. At the simplest level, operating systems and applications provide messages and logs. As systems become more complex and distributed there is a desire to filter these event messages and bring them together to a central management function.

In general, you want to design a management system that can bring together event messages from different hardware and software sources. For this reason it is important to develop a standard to define how the messages should be packaged and delivered. Fortunately, you do not have to go through this process yourself since network management architectures provide ready-made definitions. For example, SNA has generic alerts and SNMP has traps.

Traps and alerts are normally thought of as things that carry bad news. That is, some kind of alarm message. However it could equally well be good news. For example it is important to know that a resource has failed, but equally important to know that it has subsequently returned to good health.

NetView for AIX provides a variety of facilities for handling management events. With Version 4, those facilities are made stronger by a package of functions called Event Stream Enhancements. InChapter 6, "Examples of NetView for AIX Rulesets" on page 85 we show several examples that exploit event stream enhancements. Before we discuss the examples in detail, we will review the way that NetView for AIX handles events and the modifications that Version 4 has made.

5.1 Events in NetView for AIX

NetView for AIX is primarily an SNMP manager, so it is no surprise that the events that flow through it are packaged as SNMP traps. These traps come from several sources, namely:

- From SNMP agents running on distributed systems
- From status monitoring processes (which may be within NetView for AIX or on distributed Mid-Level Manager agents)
- From threshold monitoring processes (which may be within NetView for AIX or on distributed ″smart″ agents, such as IBM Systems Monitor or RMON agents)

Normally, SNMP nodes do not offer much unsolicited event information and therefore the number of events from the latter two sources is far greater than the number of traps received directly from SNMP agents.

5.1.1 A Review of SNMP Traps

The trap is the only SNMP protocol data unit (PDU) that flows unsolicited from the agent to the manager. All the other exchanges are initiated by a request from the manager. The objective of the trap is to convey alarm information from the agent. In the original conception of SNMP, the trap was expected to be used to tell the manager of a problem but not to carry fully-detailed information. The rationale for this is that the manager can determine the details using SNMP Get requests. In fact the structure of the trap allows it to be expanded easily, and some agents use traps to send a great deal of information.

Figure 55 shows the internal structure of the SNMP trap.

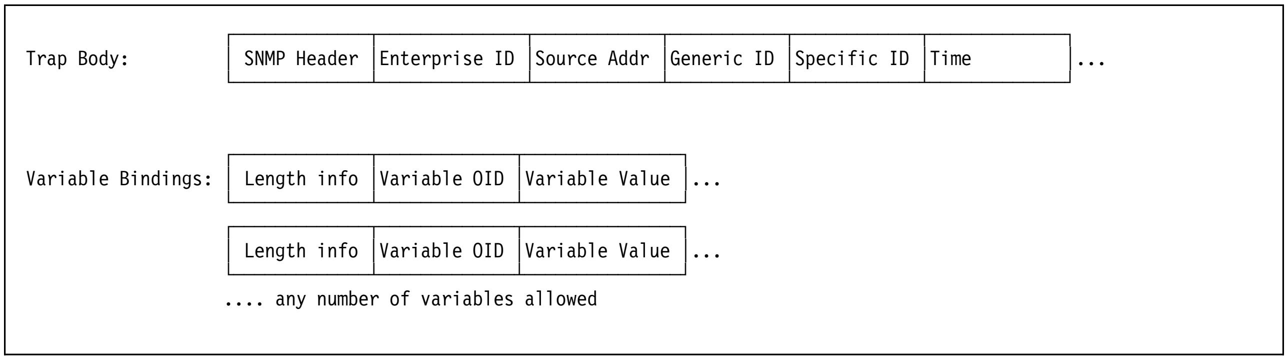


Figure 55. SNMP Trap Structure

You should note the following characteristics of this format:

- The Generic trap ID field identifies the basic type of trap. SNMP defines IDs 0-5 for specific purposes (such as Agent Warm Start and Authentication Error). Generic ID 6 is reserved for Enterprise Specific traps. Most of the traps that you see in NetView for AIX fall into this category.
- For enterprise specific traps, the combination of Enterprise ID and Specific trap ID uniquely defines the type of trap. The Enterprise ID is a MIB object ID, usually from the private part of the MIB tree. So, for example, traps originating from NetView for AIX itself have an enterprise ID of .iso.org.dod.internet.private.enterprises.ibm.ibmProd.netView6000 (or .1.3.6.1.4.1.2.6.3). The specific trap ID is just an integer. The manufacturer of the device that originates the trap is responsible for defining what different specific IDs mean under their enterprise ID.
- A trap may include one or more items of variable data. Each item is packaged as a combination of a MIB object ID to identify what type of data it is plus the value of the data itself.

5.1.2 Event Flow in NetView for AIX

There are several daemons and other processes involved in processing events as they flow through NetView for AIX. Figure 56 on page 81 shows the relationship between these processes.

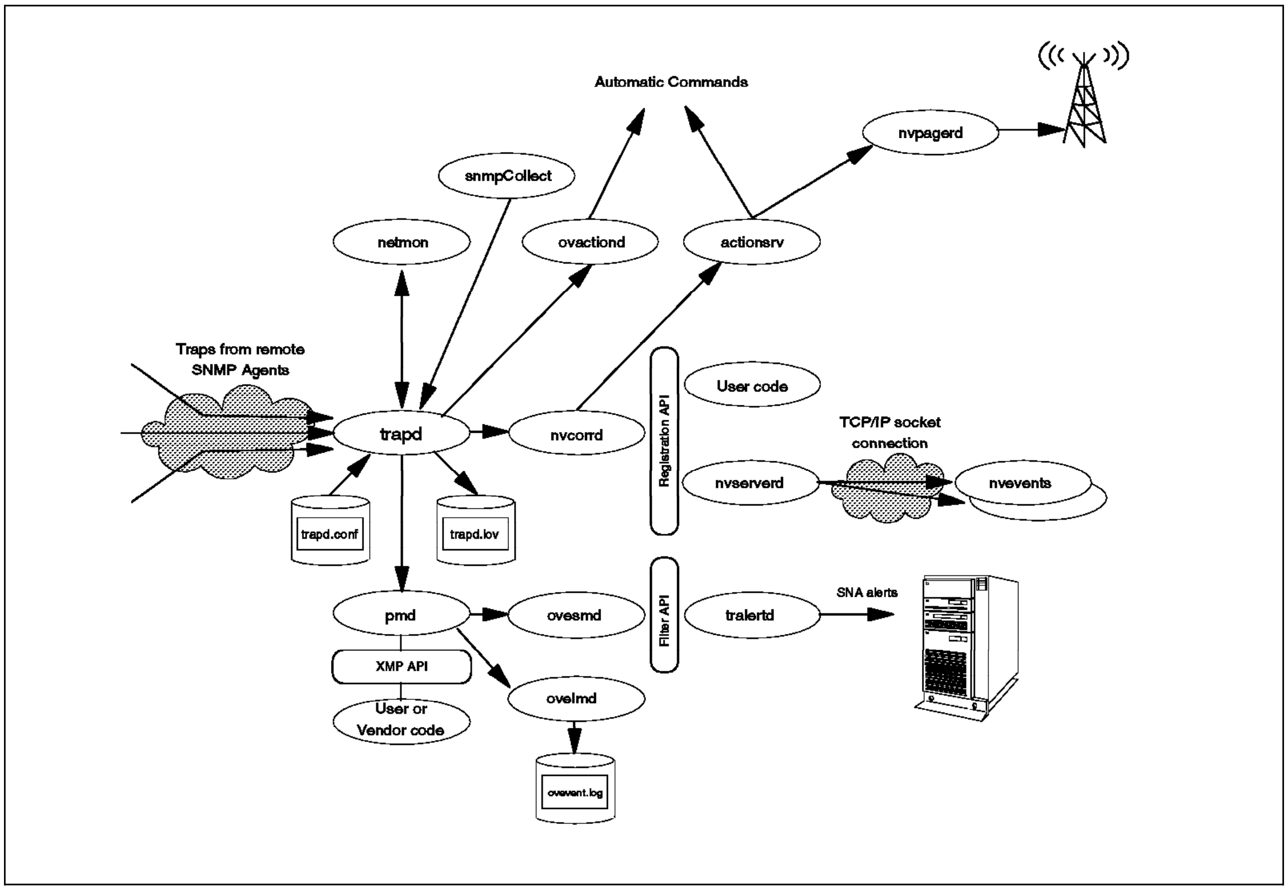


Figure 56. NetView for AIX Event Processing Flow

In this book we will not be dealing with every aspect of the event processing system, only with the event stream enhancements of NetView for AIX Version 4. However, it is useful to briefly summarize what each component does:

trapd
This daemon receives SNMP traps from remote SNMP agents and also events that are generated internally by other NetView for AIX components. Notable sources for such events are netmon (which generates events each time it detects some change in any IP resource status) and snmpCollect which generates events whenever a MIB threshold is exceeded or re-armed. The trapd daemon processes events based on records in the /usr/OV/conf/C/trapd.conf file. The things defined here are:

- Text format for the event - what is to appear on the event display and in the log.
- Category and severity of the event.
- Whether to display the event or only to log it. The trapd daemon logs all events it receives in /usr/OV/log/trapd.log.
- Whether to perform some automated action when the event occurs.

ovactiond
This daemon executes the automatic commands defined in trapd.conf. Any AIX program or shell script can be invoked.

pmd
This daemon, the *postmaster daemon* provides the XMP API for OSI CMIP applications to use. It merges events generated through XMP with the trapd event flow.

ovesmd This is the *sieve agent*. It provides a filtering capability so that applications can register with it to receive a filtered event flow. This function has been superseded by the event stream enhancements application registration facility, but it is still used by the tralertd daemon to generate SNA alerts from NetView for AIX events.

nvcorrd This daemon is the *correlation engine*. It processes the event stream from trapd and applies the correlation rules to it. This is the heart of the event stream enhancements, and we will discuss it in detail in 5.1.3, "What Are NetView for AIX Version 4 Event Stream Enhancements?."

nvserverd/nvevents nvevents is the display application that each user runs to display events as event cards or in a list form. NetView for AIX Version 4 allows users to be distributed to different client machines. nvserverd is the daemon on the NetView for AIX server that routes the event flow to the nvevents clients and also provides communication between them.

actionsvr/nvpagerd If a ruleset being processed by nvcorrd calls for a command to be run, this daemon executes it. Any AIX program or shell script can be automatically executed in this way. Invoking a call to an alphanumeric pager is a special case of a ruleset action. It is the nvpagerd daemon that initiates the call.

5.1.3 What Are NetView for AIX Version 4 Event Stream Enhancements?

At the start of this chapter we said that the first thing we want to do is to bring together network and system events in one place in a consistent format. Once this has been done, we want a management application to work for us in categorizing, filtering and adding value to the events.

We have seen how NetView for AIX achieves this, by putting external and internal events into SNMP trap format and submitting them to a sequence of processes. The processes available in NetView for AIX Version 3 (trapd, ovactiond, the ovesmd filter mechanism and nvevents) provide several facilities for handling the event stream:

- Assignment of priority and additional information
- Suppression of unwanted events, by sending them only to the log
- Automation: executing a command whenever a specific event occurs
- Viewing facilities to allow different selections of events to be seen, based on the data within them

With Version 4, the design objective was to make this more sophisticated and powerful, as follows:

- Correlation between different events, or the ability to treat two related events as one trigger
- Read and write access to additional information, external to the event stream itself (for example, the object database, MIB data, global variables)
- An override capability so that event severity and node status may be dynamically modified
- For all of these features, the ability to trigger on and have access to *any* of the information carried within the event

The NetView for AIX Event Stream Enhancements (ESE) are the outcome of this design objective. Figure 57 on page 83 shows the components of ESE and the connections between them.

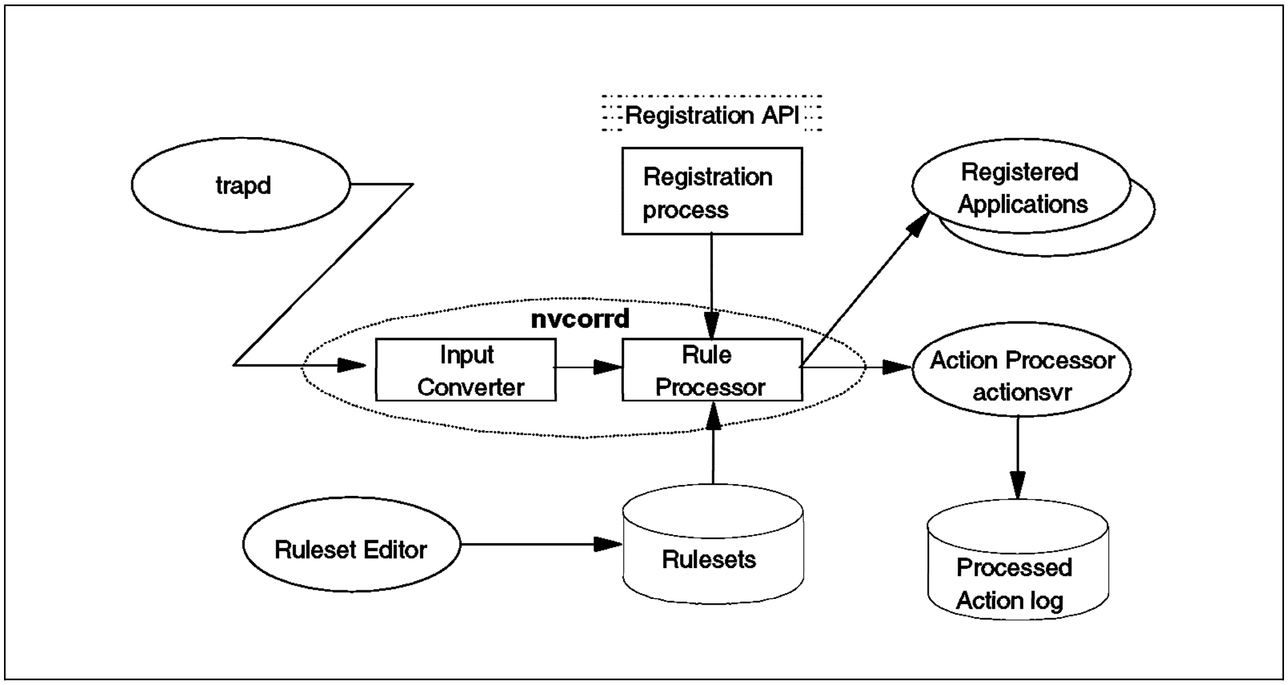


Figure 57. Components of the Event Stream Enhancements

In Figure 57 the correlation engine is subdivided into two parts, the input converter and the rule processor. These two functions are both performed by the nvcorrd daemon. The nvcorrd daemon analyzes the incoming event stream using a kind of program called a *ruleset*. These rulesets may be thought of as road maps, with several routes through them. As an event progresses along any of the routes it has to pass tests at a series of checkpoints along the road. In ESE these checkpoints are called *decision nodes*.

As well as decision nodes, ESE defines a set of *action nodes* which, as their name implies, cause some action to be taken. An event continues along a given route in the ruleset until it reaches a test (decision node) that it cannot pass or until it comes to the end of the route. As we have said, there may be many routes through the ruleset and nvcorrd will attempt to send every event that it receives along all of them. Clearly it is best for performance reasons to place the most restrictive decision node at the beginning of each route through the ruleset.

Each ruleset is stored as a single file containing definitions for the nodes and the connections between them. However, you do not need to understand the format of these files. ESE includes a powerful graphical ruleset editor which allows you to draw the ruleset as a flow diagram and then save it in the ruleset file. We show several worked examples using the ruleset editor in Chapter 6, “Examples of NetView for AIX Rulesets” on page 85.

An application that wishes to use ruleset processing has to do two things:

- Register itself and the ruleset it wants to be processed
- Wait for events to be forwarded to it

ESE has an API which provides these functions, however in the current release of NetView for AIX the API is not published. The only applications that use ESE currently are:

- nvevents (the event display application). In fact it is the server component, nvserverd, that opens the interface (see Figure 56 on page 81). The nvevents application allows you to create dynamic event workspaces with rulesets applied to them.
- actionsvr. This daemon executes automatic commands when they are invoked within a ruleset by an Action or Pager node. In fact, actionsvr performs this function for any application, so if you activate a dynamic event workspace and the ruleset includes an Action node, it will be executed by actionsvr. You can also register rulesets that *only* perform automated actions by updating a configuration file.

Chapter 6. Examples of NetView for AIX Rulesets

In this chapter we will explore the capabilities of NetView for AIX Version 4 event stream enhancements through the use of examples. We will show how to operate the Ruleset Editor and construct flow diagrams called rulesets. Our intention is to demonstrate the wide range of functions offered. In later chapters (Chapter 4, "The Collection Facility" on page 53 and 6.3, "Combining ESE Rulesets" on page 166) we will show how rulesets can be combined together and how they can be used in conjunction with other features of NetView for AIX Version 4.

Each example is described in a step-by-step format. However, to avoid repetition we cover the mechanics of ruleset creation in more detail in the first examples than in later examples. First we will give a short introduction to the Ruleset Editor. You will find the worked examples themselves in 6.2, "Examples Using the Ruleset Editor" on page 91.

6.1 Understanding the Ruleset Editor

The Ruleset Editor is available by selecting **Tools→Ruleset Editor** from the NetView for AIX menu bar, or by dragging the Ruleset Editor icon from the tool bar and releasing it anywhere on the screen. Two windows will appear on the screen: The Template Window and the Ruleset Work Area, as shown in Figure 58 on page 86.

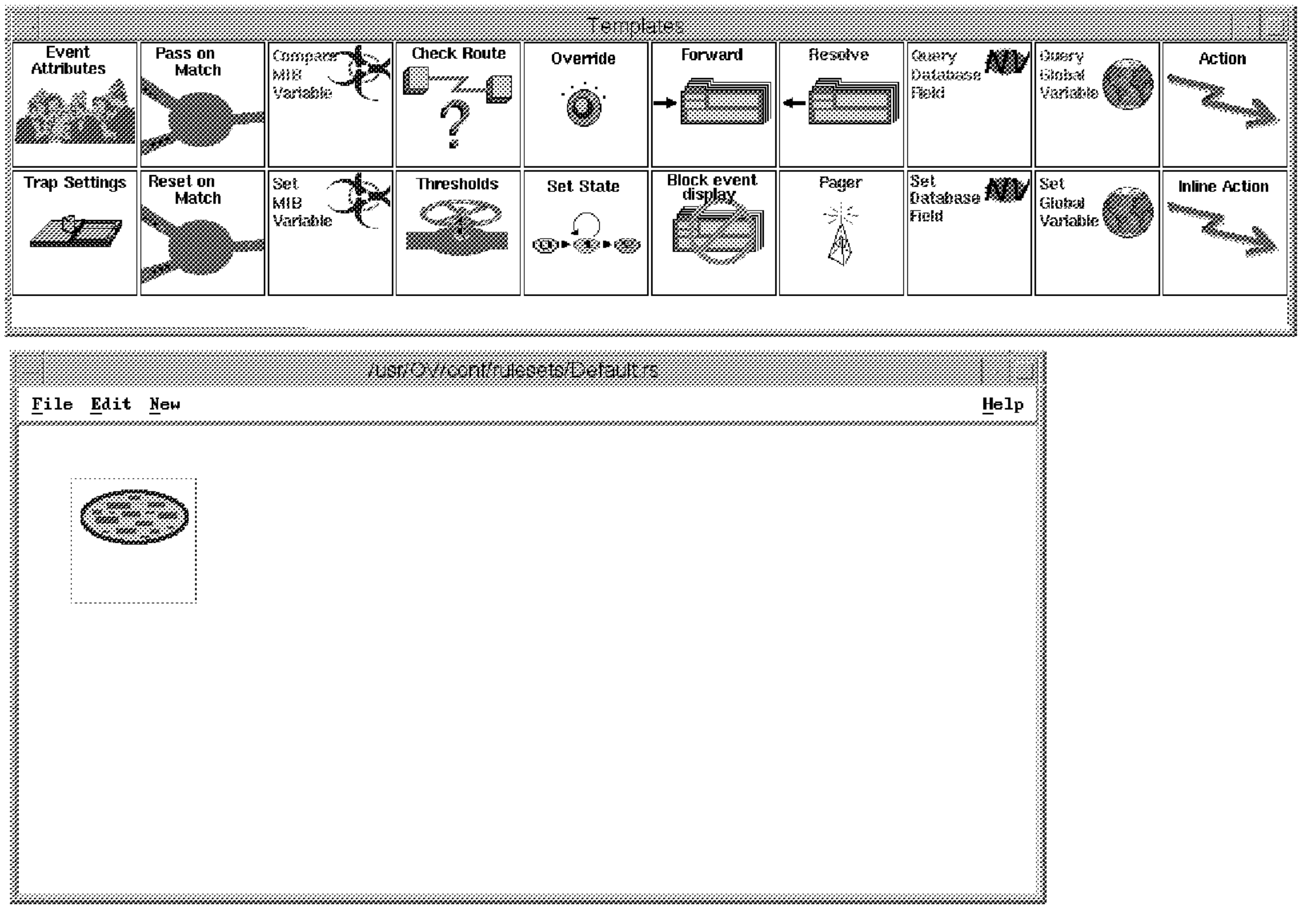


Figure 58. The Ruleset Editor's Initial Appearance

The Template window contains all the possible templates of nodes what can be added to the Ruleset Work Area. To add a node to the Ruleset area, simply drag a template from the Template window into the work area.

A brief description of each type of template is listed inTable 4 .

Table 4 (Page 1 of 3). Ruleset Editor Templates. Decision nodes control whether an event proceeds further into the ruleset. Action nodes invoke some asynchronous or synchronous action.

Template	Node Type	Description
Action	Action	Specifies the action to be performed when an event is forwarded to this node. For example, you could use this node to execute the /usr/OV/bin/ovxecho command to display a dialog window. The action defined is performed by the actionsvr daemon, as we discussed in 5.1.3, "What Are NetView for AIX Version 4 Event Stream Enhancements?" on page 82.
Block Event Display	Action	Causes the event not to be forwarded (if the default action is to forward it).

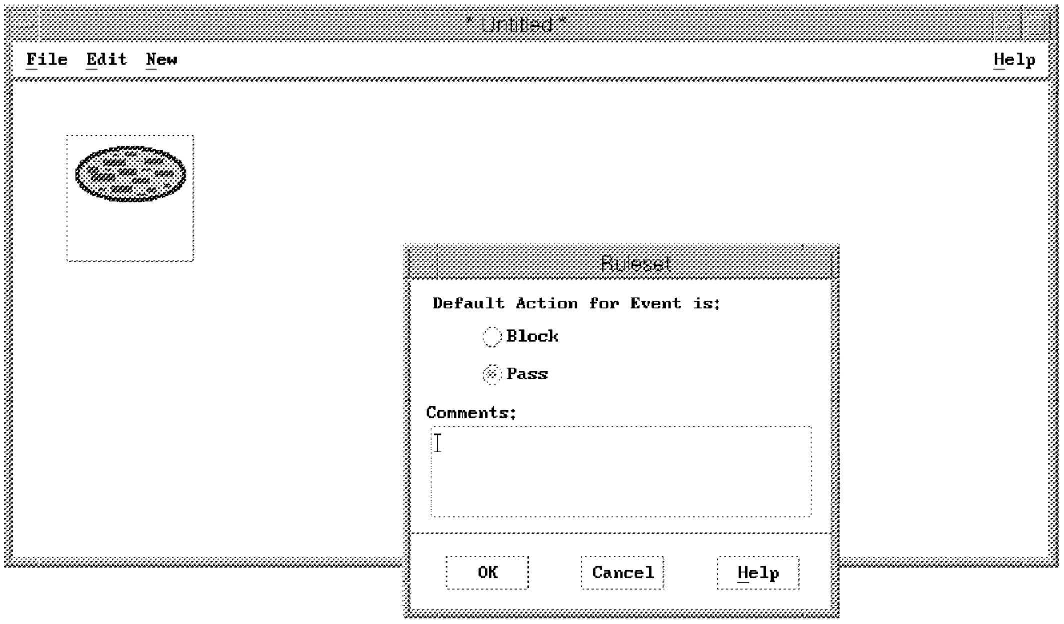
Table 4 (Page 2 of 3). Ruleset Editor Templates. Decision nodes control whether an event proceeds further into the ruleset. Action nodes invoke some asynchronous or synchronous action.

Template	**Node Type**	**Description**
Check Route	Action	Checks for communication between two network nodes and forwards the event based on the availability of this communication. For example, you can use this node to check the path from the Manager to a device before forwarding a node down trap.
Event Attributes	Decision	Compares any attribute of the incoming event to a literal value. For example, you can use this node to check for events generated by a particular device.
Forward	Action	Forwards the event to applications that have registered to receive the output of the ruleset. For example, when the registered application is nvevents, you must use the Forward node if you want to display the event.
Inline Action	Action	Specifies an action to be performed. Unlike the Action node, which always executes the action under the actionsvr daemon, the Inline Action executes under the main ruleset processing daemon, nvcorrd. Subsequent ruleset nodes wait for the action to complete (or for a timeout to expire).
Override	Action	Overrides the object status or severity assigned to a specific event and updates the Events Display application. For example, you can use this node to change the severity to Major when a node down event is received for a router. Use this node with the query database field node to override status or severity for specific device types.
Pager	Action	Issues a call to a pager that has been defined in a NetView for AIX user profile.
Pass on Match	Decision	Compares attributes between two events. You can use this node to check for two events that have something in common, for example, two events generated by the same node in the network.
Query Database Field	Decision	Compares a value from the NetView for AIX object database to a literal value or to a value contained in the incoming event. For example, you can use this node to check if the originating device is a router.
Query Global Variable	Decision	Queries the value of the global variable that has been previously set using the Set Global Variable node.
Reset on Match	Decision	Delays an event until either a timer has expired or an event with matching attributes occurs. You can use this node to check for two events that have something in common, for example, two events generated by the same node in the network.
Resolve	Action	Forwards a message to all registered applications indicating that a previous event has been resolved. You can use this node to delete an event card from the events display application when an subsequent event is received.
Set Database Field	Action	Sets the value of any NetView for AIX object database field.

Table 4 (Page 3 of 3). Ruleset Editor Templates. Decision nodes control whether an event proceeds further into the ruleset. Action nodes invoke some asynchronous or synchronous action.

Template	**Node Type**	**Description**
Set Global Variable	Action	Sets a variable for use within the ruleset itself. For example, use this node to set a flag whose value will be checked later in the ruleset using the query global variable node.
Set MIB Variable	Action	Issues an SNMP SET command to set the value of a variable in the MIB representing any network resource. For example, you can use this node to dynamically change the configuration of a LAN hub device.
Set State	Action	Sets the correlation state of an object in the NetView for AIX object database. The current state is updated in the corrstat1 field in the object database, and the previous value in the corrstat1 field is moved to the corrstat2 field. This process continues until the current state and as many as four previous states are stored in the object database. You can view the correlation state by selecting the object and then selecting the Display Correlation Status option from the context menu.
Thresholds	Decision	Checks for repeated occurrences of the same trap or of traps with an attribute in common. You can use this node to forward an event after receiving the specific number of the same event received within a specific time period. Use this node with the Trap Settings node to identify a specific trap number.
Trap Settings	Decision	Specifies a specific trap to be processed and is identified by a pair of generic and specific trap numbers.

The ruleset work area is like an canvas upon which you can draw your ruleset. When it is first initialized it contains the Event Stream node (which looks exactly like a pizza). This node represents all events passed to nvcorrd by the trapd daemon. If you double-click on this node you will be prompted for the default behavior for the ruleset you are constructing (see Figure 59 on page 89). If you set this to Pass, all events will be forwarded to the registered application unless you place a Block Event Display node to prevent them.



Default Action for Event is:

Block [✓ Pass]

Comments:

Figure 59. Defining Default Event Behavior

The Ruleset work area window provides you with several menu bar functions that are useful in editing your rulesets. There are 3 items: File, Edit, and New.

The menu items available under File are:

New	Create a New RSFile.
Insert	Read an existing ruleset and insert it into the current ruleset. We will describe this option in more detail in 6.3, “Combining ESE Rulesets” on page 166.
Open	Open a previous created RSFile.
Save	Save a RSFile.
Save as	Save a RSFile with a new name.
Delete	Delete a RSFile.
Exit	Exit from Ruleset Editor.

The menu items available under Edit are:

Connect two Nodes	Connect two different nodes with an arrow.
Delete node	Select a node and delete it.
Delete line	Select a line between two nodes and delete it.
Refresh Layout	Redraw the ruleset and rearrange the nodes.
Unselect ...	Unselect all the graphical symbols.
Reset Cursor	Reset the cursor layout (useful if you select an operation and then decide not to complete it).
Delete Selected ..	Delete all the selected graphical symbols.

The menu items available under New are another means by which the templates can be accessed. Each menu item corresponds to a template which is also available in the Templates window.

6.2 Examples Using the Ruleset Editor

There are eleven ruleset examples in this section. Table 5 summarizes them.

Table 5. Summary of the Ruleset Examples

Example	Ruleset Nodes Used	Description
6.2.1, “Clearing Outstanding Events via Correlation” on page 92	Trap Settings, Pass on Match, Resolve, Block Event Display	This example looks for interface and node Up events that match a previous Down event. If a match is found, the Down event is removed from the nvevents display and the Up event is suppressed.
6.2.2, “Suppressing Events by Setting Thresholds” on page 101	Trap Settings, Thresholds, Forward, Action	This example checks for repeated performance threshold events and suppresses them if they are occurring at less than a given frequency.
6.2.3, “Using Thresholds in Combination with Correlation” on page 108	Trap Settings, Thresholds, Pass on Match, Forward, Action	This example performs suppression in a similar way to the previous example, *unless* two types of performance error are being reported by the same managed node.
6.2.4, “Automated Paging and E-Mail Notifications” on page 112	Trap Settings, Event Attributes, Pager, Action	This example checks for a node down event from a specific router and generates a paging request and an e-mail message.
6.2.5, “Using Traps to Override Status Color and Severity” on page 118	Trap Settings, Override, Query Database Field	This example overrides the status (and hence the symbol color) of a node as a result of a threshold trap being received. It also shows how to override the severity of an event based on the contents of an object database field.
6.2.6, “Setting Correlation States” on page 124	Trap Settings, Event Attributes, Set State, Query Database Field, Override	This example uses the correlation state fields in the object database to keep track of an automatic recovery process.
6.2.7, “Setting Database Fields” on page 133	Trap Settings, Event Attributes, Set Database Field, Override, Forward	This example changes the value of an object database field based on the contents of traps sent by NetFinity.
6.2.8, “Setting Global Variables” on page 140	Trap Settings, Set Global Variable, Query Global Variable, Action, Forward, Thresholds	This example determines whether to display a trap from an application based on whether the underlying communications are active.
6.2.9, “Setting MIB Variables” on page 147	Trap Settings, Set MIB Variable, Forward, Action	This example uses a SNMP set request to modify a Systems Monitor agent polling frequency when a threshold is exceeded.
6.2.10, “Using Rulesets to Supplement Event Capabilities of Another Manager” on page 151	Trap Settings, Event Attributes, Pass on Match, Action, Forward	This example shows how a ruleset can be devised to cause automated interactions with another management platform, in this case, NetFinity.
6.2.11, “Suppressing Events for Interfaces That Are Administratively Down” on page 162	Trap Settings, Query Database Field, Inline Action, Block Event Display	This example shows how to use an Inline Action as a way to test for information that is not in the trap and thus control the process flow.

In each example we show a flow diagram to illustrate the processing that we want the ruleset to perform. The flow diagrams have been added for clarity, but we found it a very useful design aid to draw out such diagrams whenever we were contemplating creating a new ruleset.

6.2.1 Clearing Outstanding Events via Correlation

In most network operation centers, operators are not interested in problems that have already been solved. At the very least, they do not want to mistake solved problems with outstanding problems. ESE′s correlation feature provides an easy way to clear solved problems from NetView for AIX′s event display automatically. In this example we will show how you can clear a Node Down event from the event display if a Node Up event arrives for the same node in a defined interval of time. We will take the same actions for the Interface Down and Interface Up traps.

Figure 60 displays a flow of what we want to obtain with this ruleset.

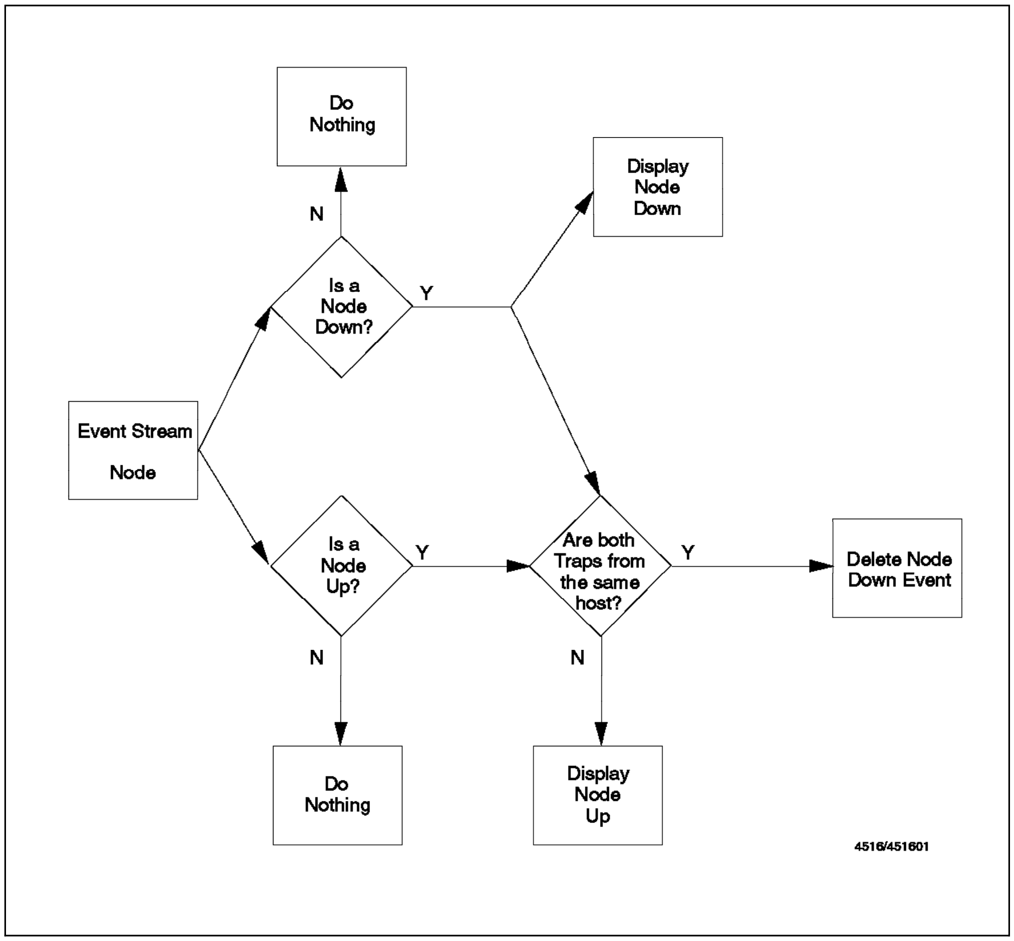


Figure 60. Flowchart of Correlation Example

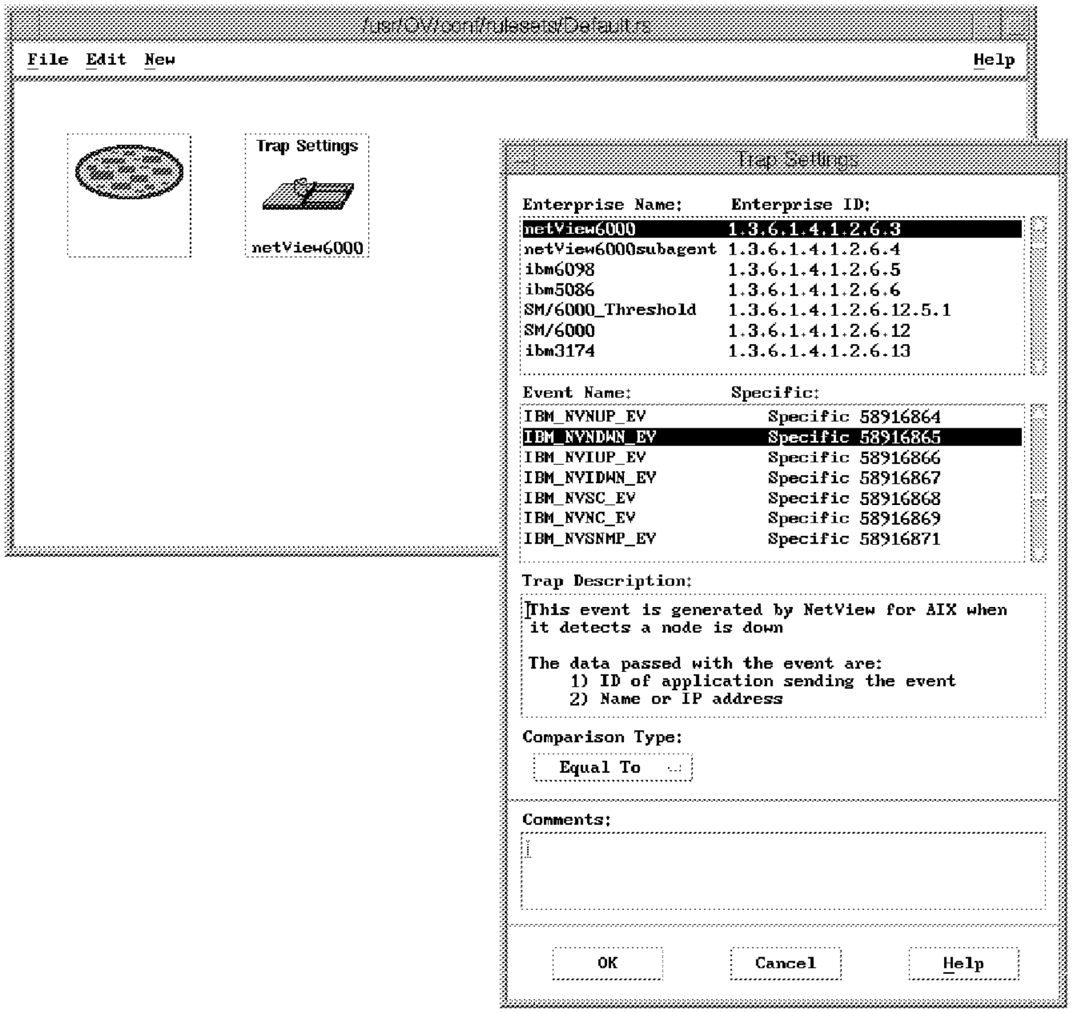
6.2.1.1 Constructing the Ruleset

1. Before we add other nodes we will want to configure the default behavior. Do this by double-clicking the event stream node as shown in Figure 59 on page 89. We want all events to appear, unless our ruleset explicitly prevents them, so we choose the default setting of Pass.

2. The next step is to identify the first of two traps that we wish to correlate. Thus, we will drag the **Trap Settings** icon and release it on the ruleset window. Another window appears (as shown in Figure 61 on page 93) and in this new window we will identify the Node Down trap by selecting the following settings:

Enterprise Name	netview6000
Event Name	IBM_NVDWN_EV
Specific	Specific 58916865

The trap description appears when you choose the event name from the list of defined events.



```
Enterprise Name:     Enterprise ID:
netView6000          1.3.6.1.4.1.2.6.3
netView6000subagent  1.3.6.1.4.1.2.6.4
ibm6098              1.3.6.1.4.1.2.6.5
ibm5086              1.3.6.1.4.1.2.6.6
SM/6000_Threshold    1.3.6.1.4.1.2.6.12.5.1
SM/6000              1.3.6.1.4.1.2.6.12
ibm3174              1.3.6.1.4.1.2.6.13

Event Name:          Specific:
IBM_NVNUP_EV              Specific 58916864
IBM_NVNDWN_EV             Specific 58916865
IBM_NVIUP_EV              Specific 58916866
IBM_NVIDWN_EV             Specific 58916867
IBM_NVSC_EV               Specific 58916868
IBM_NVNC_EV               Specific 58916869
IBM_NVSNMP_EV             Specific 58916871

Trap Description:
This event is generated by NetView for AIX when
 it detects a node is down

 The data passed with the event are:
      1) ID of application sending the event
      2) Name or IP address

Comparison Type:
  Equal To

Comments:
```

Figure 61. Identifying the Node Down Event

3. Click on **OK** to complete the first node of the ruleset.
4. Now connect the Event Stream node to the one you have just created by selecting Edit→Connect Two Nodes from the menu bar. Your mouse pointer will change to a connection icon. Now click on the Event Stream node first and then on the Trap Settings node (the order is important, because the direction of flow is from the first to the second node selected).
5. Now we have to add the node for the Node Up Event in the same way. To accomplish this, first click on the Event Stream node and then repeat steps 1-3, but this time use the following settings:

Enterprise Name	netview6000
Event Name	IBM_NVNUP_EV
Specific	Specific 58916864

Because we first selected the Event Stream node this new node should be automatically connected to it. Hence, we obtain a ruleset like that displayed in Figure 62 on page 94.

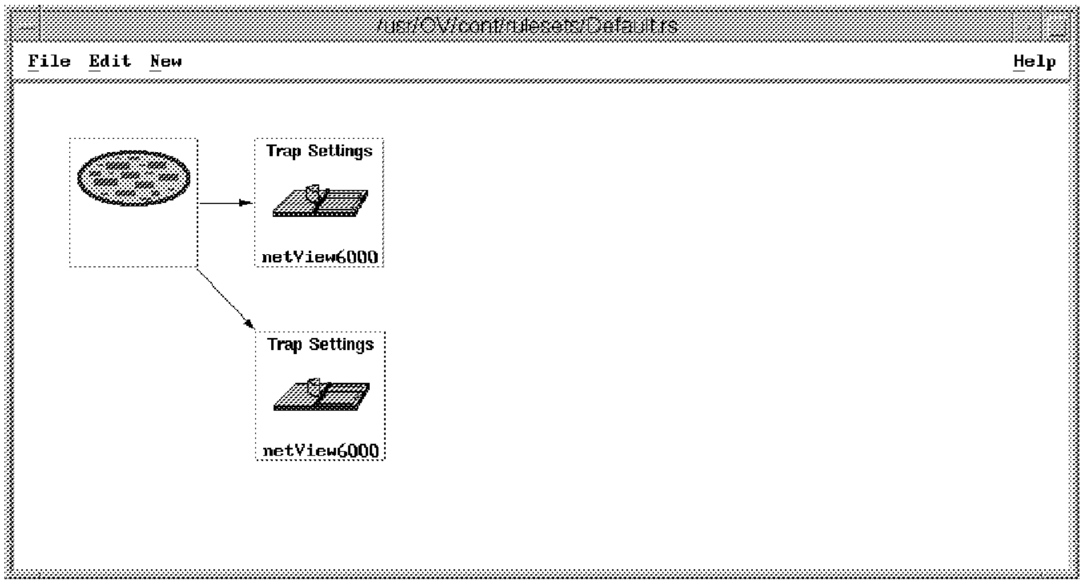


Figure 62. Adding the Trap Settings Node for the Node Up Event

Automatic Node Connections

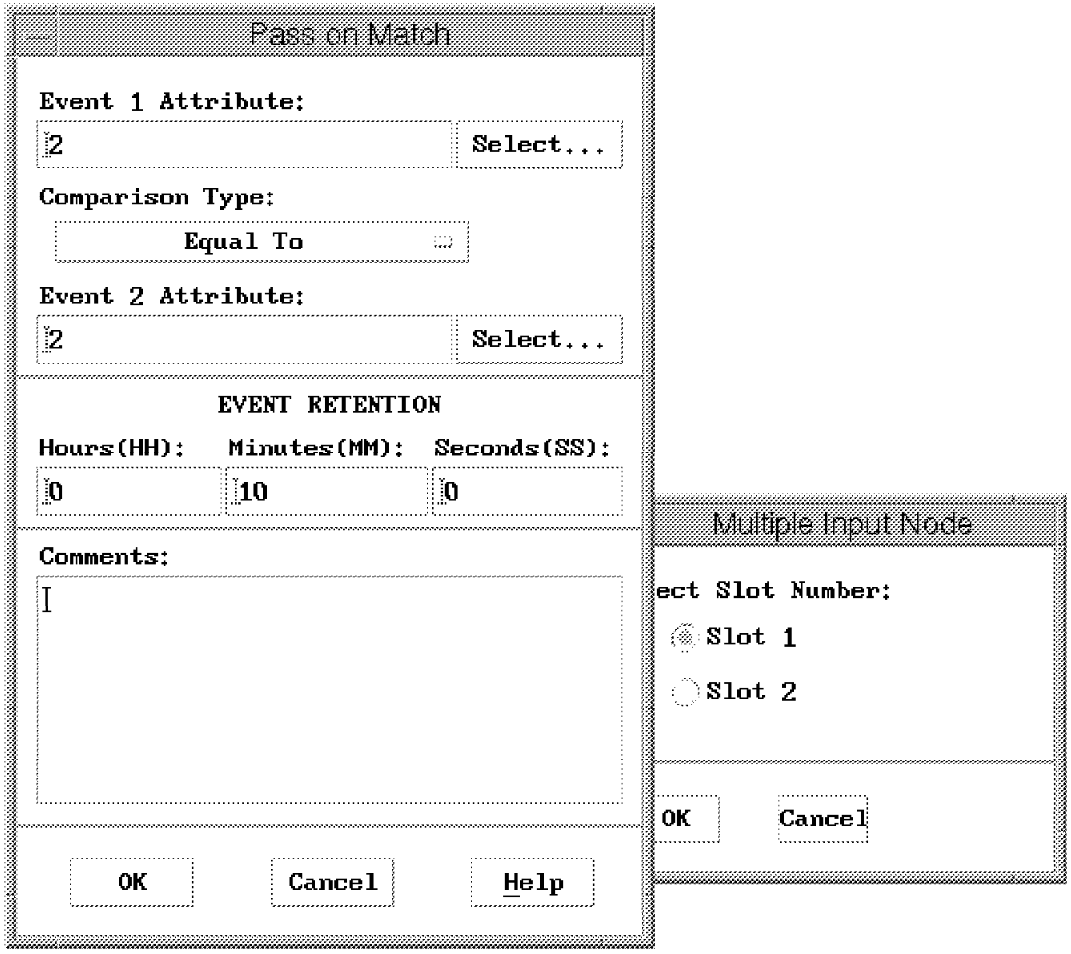
In this step we saw that the ruleset editor automatically connected two nodes together with an arrow. If you have an existing node selected in the editor, it will add an arrow from that node to the node you are currently adding. This makes your job easier because you have one less step to perform. However, it may happen that you have a node selected without intending to. In that case, you will get a connection added that you did not want. You have to remove the incorrect connection, by clicking on **Edit** → **Delete Line** and add the correct one.

6. Now drag the **Pass on Match** icon to the work area. You will see a dialog box like that presented in Figure 63 on page 95. You may also see a dialog box titled Multiple Input Node. Ignore this second box for now and complete the Pass on Match dialog. This dialog is asking you which attributes of the first and second trap you want to compare. In this case you only want to do special processing for the second (Node Up) trap if you have previously received a Node Up trap for the same node. If this trap had come from a remote node, the attribute to compare would have been the trap *origin*. However, the Node Up and Node Down traps are in fact created by NetView for AIX itself (specifically the netmon daemon). The affected node is therefore a variable in the trap. In the dialog box, select the number **2** into both the Event 1 Attribute and Event 2 Attribute fields and leave the Comparison Type set to Equal To. This means that we want to determine if the second variable in each trap is the same. For NetView traps, the second variable contains the hostname or IP address of the node that the trap is concerned with.

How do you find out about the variables in a trap?

It is up to the sender of the trap to define how many variables to include in a trap and what data to put in them. You may have to refer to documentation from the vendor to find out these details. In the case of NetView for AIX traps you can find out what variables are contained in them from the Event Customization dialog (select **Options** → **Event Configuration** → **Trap Customization** from the menu bar).

For this example, set the Event Retention Time to 10 minutes. This means that the Pass on Match node will only be passed if a Node Up event arrives within ten minutes after the corresponding Node Down event. You can think of Pass on Match as having a short-term memory. It remembers the first event until either the matching event arrives, or the timer expires.


Pass on Match

Event 1 Attribute: 2

Comparison Type: Equal To

Event 2 Attribute: 2

EVENT RETENTION

Hours(HH): 0 Minutes(MM): 10 Seconds(SS): 0

Comments:

Multiple Input Node

ect Slot Number:

(●) Slot 1

( ) Slot 2

Figure 63. Pass on Match Attributes Dialog

Select **OK** to save the Pass on Match details.

7. Return now to the Multiple Input Node dialog box (see Figure 63). This is asking you which trap should be considered Event 1 (that is, the first to arrive) and which trap should be considered Event 2. Note that Event 1 initializes the correlation function and is the trap against which potential Event 2 traps are compared. The reason the Multiple Input Node dialog box

was invoked when you dragged the **Pass on Match** node into the editor is because of the automatic connection feature we described above.

For this example, you want to specify the Node Down trap as being Event 1, so you will select **Slot 1** when you connect Node Down to the Pass on Match node and **Slot 2** when you connect Node Up. We had Node Up selected in this case, so we should select **Slot 2** and then use Edit→Connect Two Nodes to connect Node Down to Slot 1. Your ruleset should now look like that shown in Figure 64.

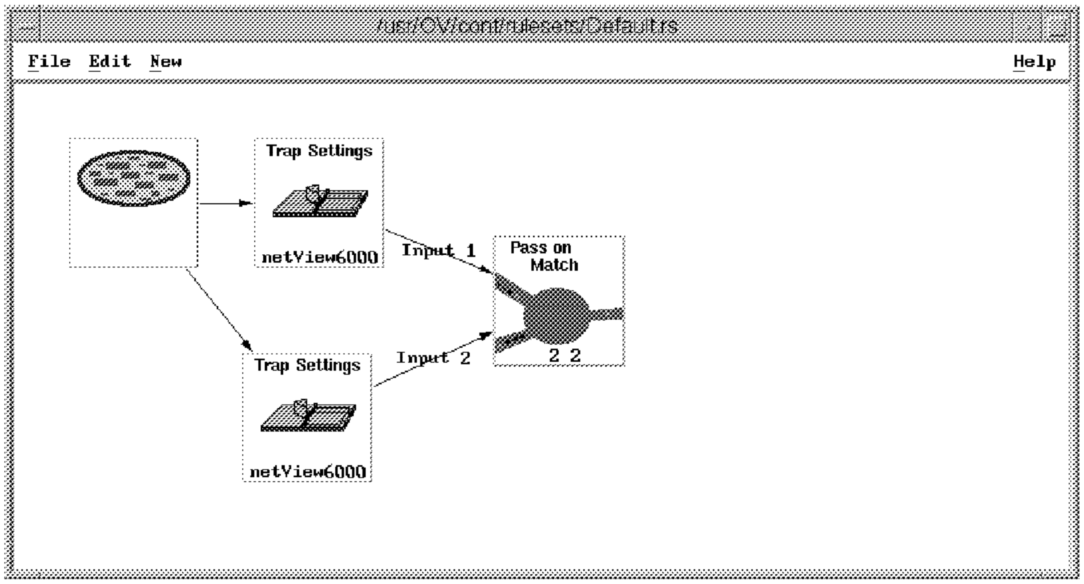


Figure 64. After Adding the Pass on Match Node

8. Now drag the **Resolve** node into the ruleset window and connect it to the Pass on Match node. This node will act to remove the initial Node Down trap from the workspace if an event passes the correlation conditions (that is, if the corresponding Node Up trap arrives before the end of the ten minute retention period).

9. Finally, we must decide what to do with the Node Up event. When we started we set the default behavior to pass events, so the Node Up will normally always appear. As we are removing clutter from the event display by removing the Node Down event, it seems reasonable also to not display the Node Up (unless they occur more than ten minutes apart). We do this by dragging the **Block Event Event Display** node onto the work area and connecting it to the Pass on Match node. Your ruleset should now look like Figure 65 on page 97.

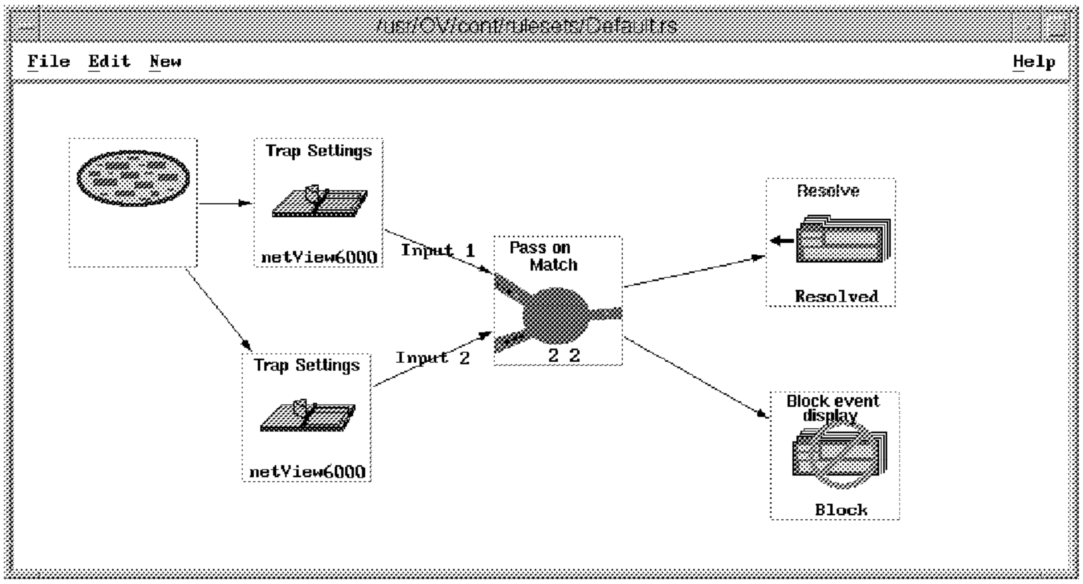


Figure 65. Adding the Resolve and Block Event Display Nodes

10. Now we can repeat the procedure with the interface down and the interface up traps. To accomplish this, repeat steps 1-8 setting the first Trap Setting node (for interface down) to:

Enterprise Name	netview6000
Event Name	IBM_NVIDWN_EV
Specific	Specific 58916867

And the second Trap Setting node (for interface up) to:

Enterprise Name	netview6000
Event Name	IBM_NVIUP_EV
Specific	Specific 58916866

Now your entire Ruleset should look like that presented in Figure 66 on page 98.

11. You should now save your ruleset via the Save As function available under the File menu option. For our purposes we named this ruleset correlation.rs (see Figure 66 on page 98).

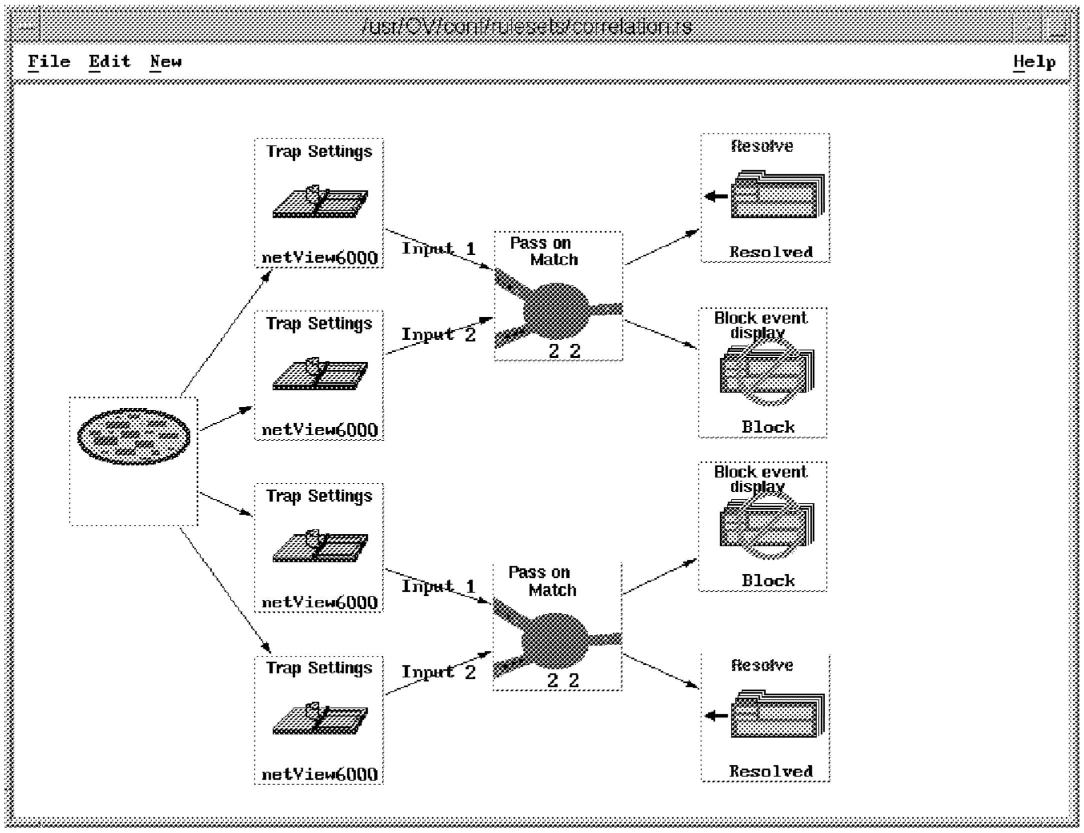


Figure 66. The Completed Correlation Example Ruleset

6.2.1.2 Testing the Ruleset

Now we can test our ruleset by completing the following steps:

1. Open a new dynamic workspace that uses our new ruleset. This can be accomplished by choosing **Create->Dynamic Workspace** from the main Events display. In the dialog that comes up, fill in the fields as follows:

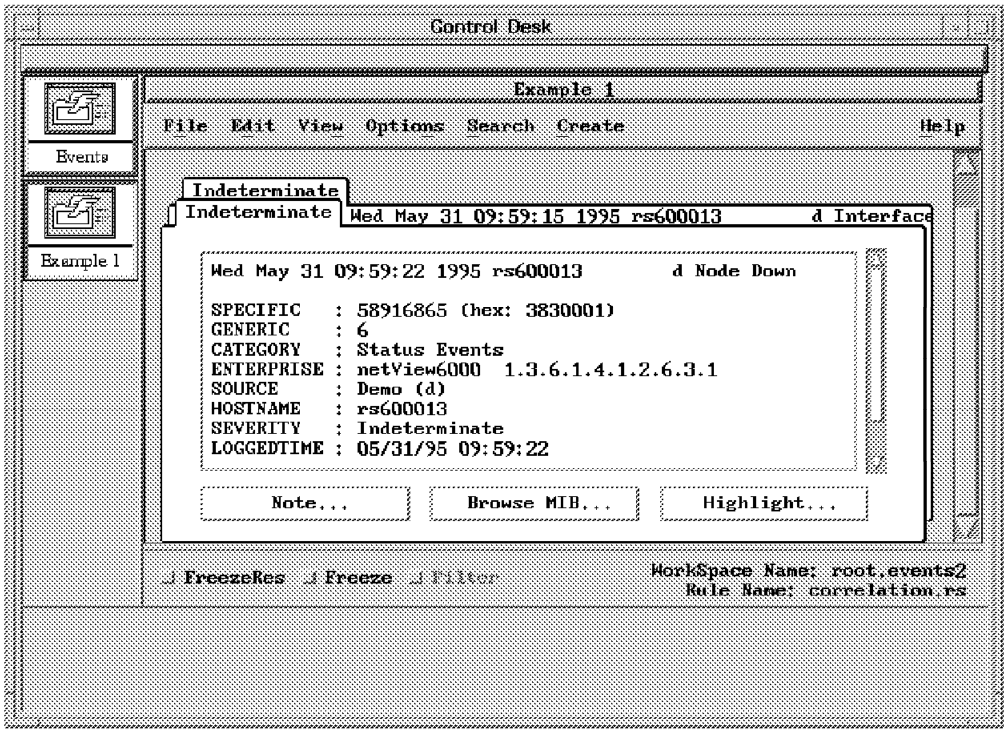
Workspace Title	Example 1
Correlation Rule	correlation.rs

   Then select **OK**. You should now see a new Dynamic Workspace.

2. Now we can send some test traps to determine if a ruleset is working. In our example, we typed the following command in order to send a node down trap for host rs600013:

   ```
   event -h rs600013 -e NDWN_EV
   ```

   You should now see a new event in your workspace like that shown in Figure 67 on page 99.



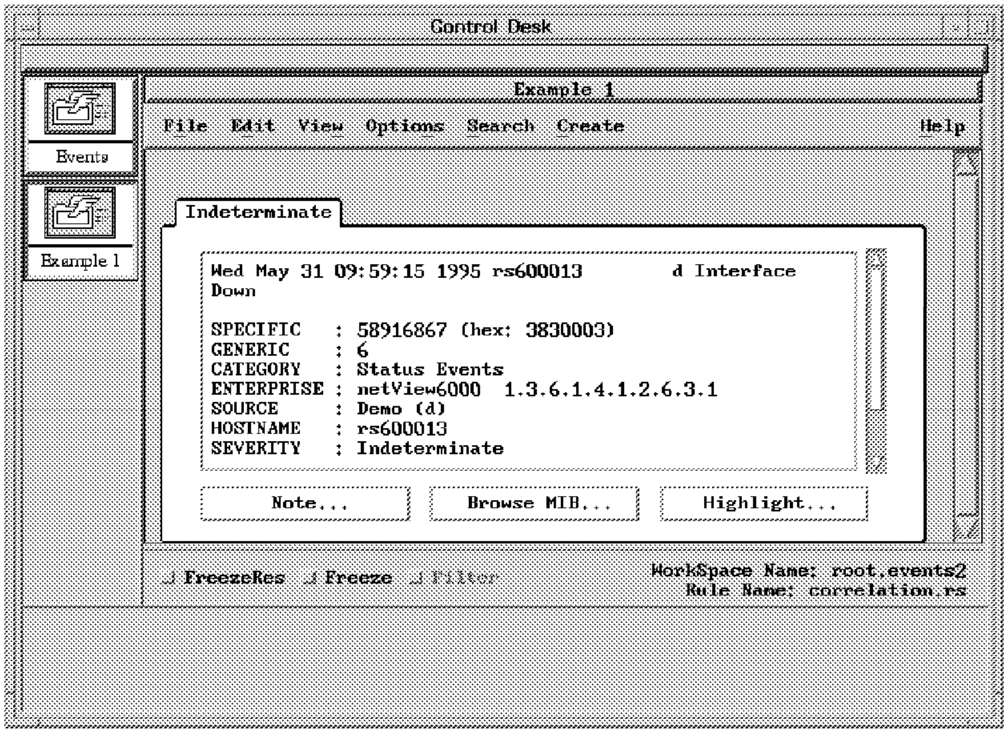
```
Wed May 31 09:59:15 1995 rs600013         d Interfac
Wed May 31 09:59:22 1995 rs600013       d Node Down

SPECIFIC   : 58916865 (hex: 3830001)
GENERIC    : 6
CATEGORY   : Status Events
ENTERPRISE : netView6000  1.3.6.1.4.1.2.6.3.1
SOURCE     : Demo (d)
HOSTNAME   : rs600013
SEVERITY   : Indeterminate
LOGGEDTIME : 05/31/95 09:59:22

WorkSpace Name: root.events2
Rule Name: correlation.rs
```

Figure 67. The Node Down Event Arrives

3. Now we can send a Node Up Event message for the same host (rs600013) by typing the following command:

   ```
   event -h rs600013 -e NUP_EV
   ```

   Immediately, the previously displayed event will disappear from our workspace like that shown in Figure 68 on page 100.


Control Desk
Example 1
File Edit View Options Search Create Help
Events
Example 1
Indeterminate
Note... Browse MIB... Highlight...
FreezeRes Freeze Filter

```
Wed May 31 09:59:15 1995 rs600013        d Interface
Down

SPECIFIC   : 58916867 (hex: 3830003)
GENERIC    : 6
CATEGORY   : Status Events
ENTERPRISE : netView6000  1.3.6.1.4.1.2.6.3.1
SOURCE     : Demo (d)
HOSTNAME   : rs600013
SEVERITY   : Indeterminate
```

WorkSpace Name: root.events2
Rule Name: correlation.rs

Figure 68. The Node Down Event Disappears After It Is Resolved

4. A similar test can be down with the Interface Up and Interface Down traps. You can test this by using the following commands:

```
event -h rs600013 -e IDWN_EV

event -h rs600013 -e IUP_EV
```

6.2.2 Suppressing Events by Setting Thresholds

A frequent source of event clutter in an event display are those traps that keep repeating themselves. You may be interested in these traps but do not need to see them as frequently as they are appearing in your display. Or, perhaps you are not interested in the circumstances that the trap is reporting unless the trap is being sent at a rapid rate.

One example of a method to filter repeated events is provided by the NetView for AIX Data Collection and Thresholding function. This is a built-in application that polls for MIB data and compares it with a threshold. Data Collection and Thresholding provides the *rearm* capability, as a method to filter out unwanted ″event noise″. Figure 69 shows how rearm works.

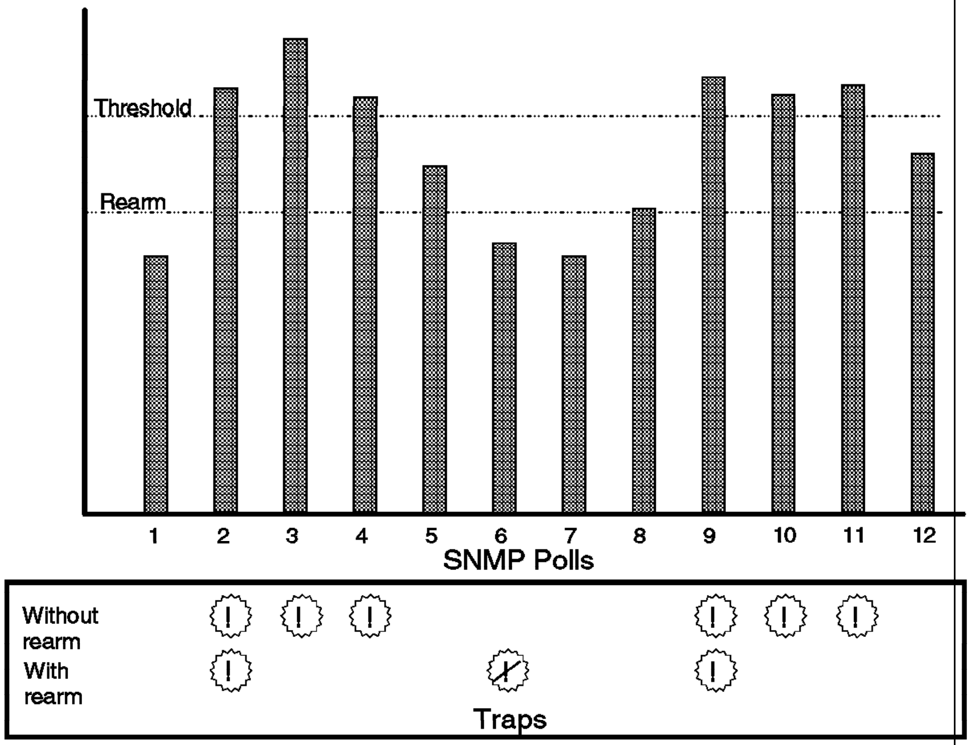
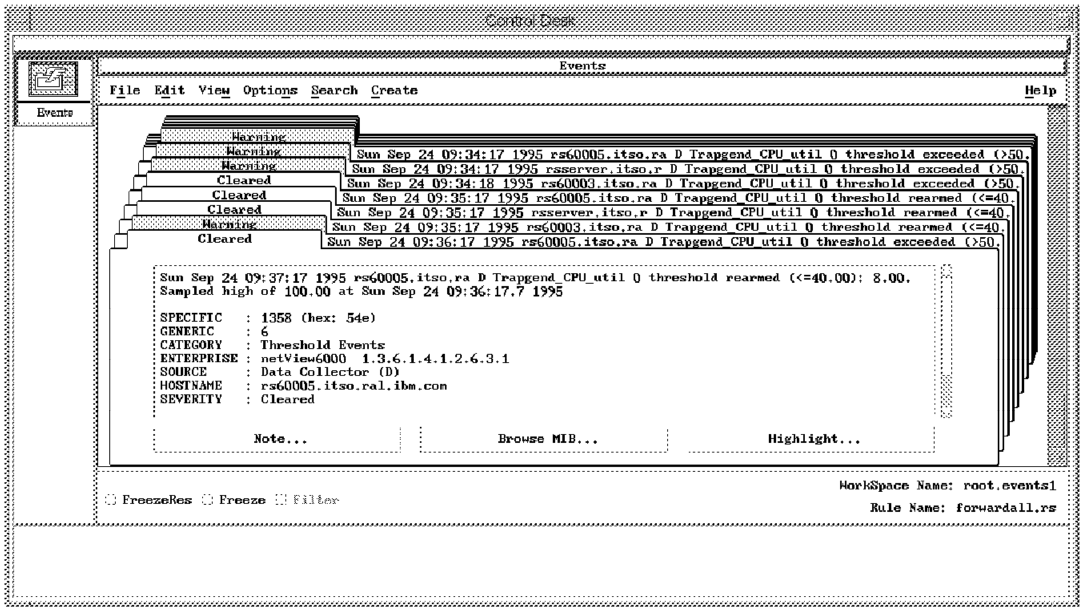


Figure 69. How the Threshold/Rearm Function Works. The rearm level is normally set a little below the threshold level. In the example above, instead of six threshold events, we would see two threshold events and one rearm event.

This rearm capability is only effective if you are monitoring counters with a high degree of consistency. If the counter varies erratically from one sample to the next you are likely to again be flooded with events, but now they will be threshold events and rearm events interspersed.

You can use a different approach to this problem by using the NetView for AIX ruleset Threshold function. This allows you to establish criteria that are based on the frequency of a trap′s arrival in a defined interval of time.

In this example, we will use Data Collection and Thresholding to poll for CPU utilization. This is a figure provided by the trapgend agent which is part of NetView for AIX (you can distribute trapgend to any RS/6000s in your network). The CPU utilization figure that trapgend gives is an instantaneous reading. It therefore tends to be very erratic and using rearm just results in a large number of threshold/rearm event pairs. See Figure 70 for an example of this.



```
Warning
Warning   Sun Sep 24 09:34:17 1995 rs60005.itso.ra D Trapgend_CPU_util 0 threshold exceeded (>50.
Warning   Sun Sep 24 09:34:17 1995 rsserver.itso.r D Trapgend_CPU_util 0 threshold exceeded (>50.
Cleared   Sun Sep 24 09:34:18 1995 rs60003.itso.ra D Trapgend_CPU_util 0 threshold exceeded (>50.
Cleared   Sun Sep 24 09:35:17 1995 rs60005.itso.ra D Trapgend_CPU_util 0 threshold rearmed (<=40.
Cleared   Sun Sep 24 09:35:17 1995 rsserver.itso.r D Trapgend_CPU_util 0 threshold rearmed (<=40.
Warning   Sun Sep 24 09:35:17 1995 rs60003.itso.ra D Trapgend_CPU_util 0 threshold rearmed (<=40.
Cleared   Sun Sep 24 09:36:17 1995 rs60005.itso.ra D Trapgend_CPU_util 0 threshold exceeded (>50.

Sun Sep 24 09:37:17 1995 rs60005.itso.ra D Trapgend_CPU_util 0 threshold rearmed (<=40.00): 8.00.
Sampled high of 100.00 at Sun Sep 24 09:36:17.7 1995

SPECIFIC    : 1358 (hex: 54e)
GENERIC     : 6
CATEGORY    : Threshold Events
ENTERPRISE  : netView6000  1.3.6.1.4.1.2.6.3.1
SOURCE      : Data Collector (D)
HOSTNAME    : rs60005.itso.ral.ibm.com
SEVERITY    : Cleared

WorkSpace Name: root.events1
Rule Name: forwardall.rs
```

Figure 70. Threshold/Rearm with Erratic Data. The CPU utilization figure is a snapshot, so it can vary wildly from one sample to the next, causing a lot of threshold/rearm events.

What we want to do is only generate an alert if the threshold is persistently high. Using a ruleset we will only display the event if it occurs five times consecutively (specifically, we will send the fifth trap that occurs inside of six minutes). We will also display a message on the screen if this trap is forwarded so that we can emphasize its arrival. Figure 60 on page 92 displays a flow of what we want to obtain with this ruleset.

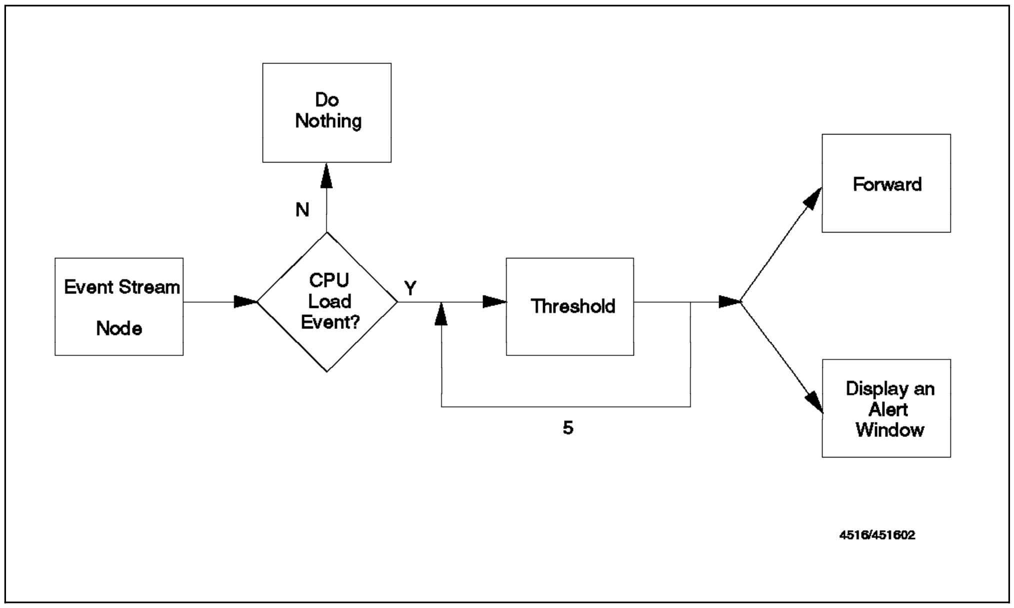


Figure 71. Flowchart of Threshold Example

Figure 72 displays the ruleset that we constructed for this example.

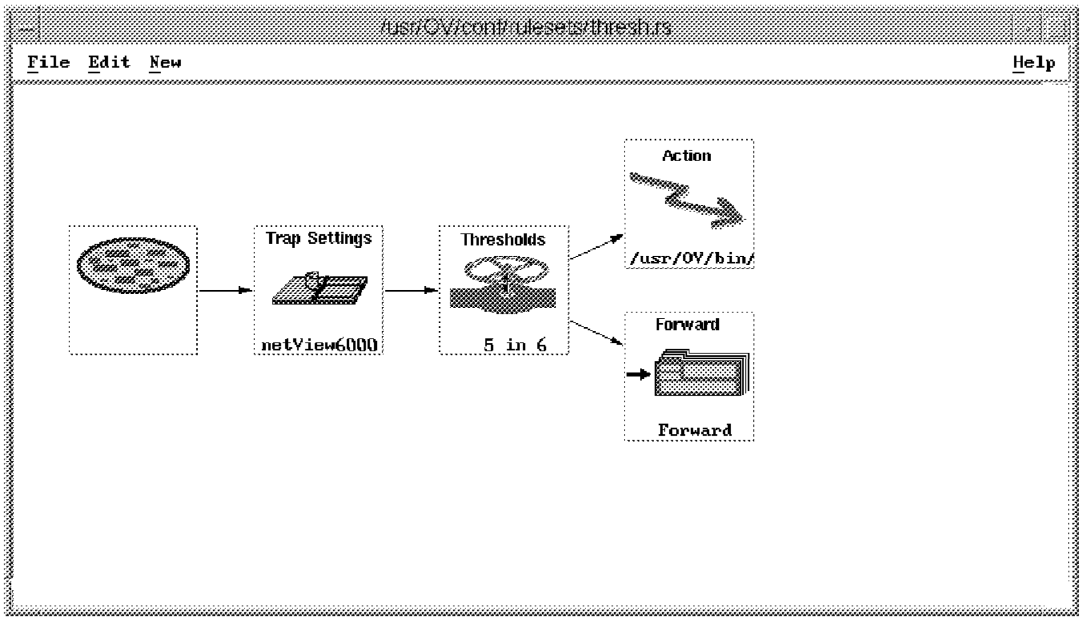


Figure 72. Basic Ruleset Using the Threshold Function

6.2.2.1 Setting up the Threshold Monitor

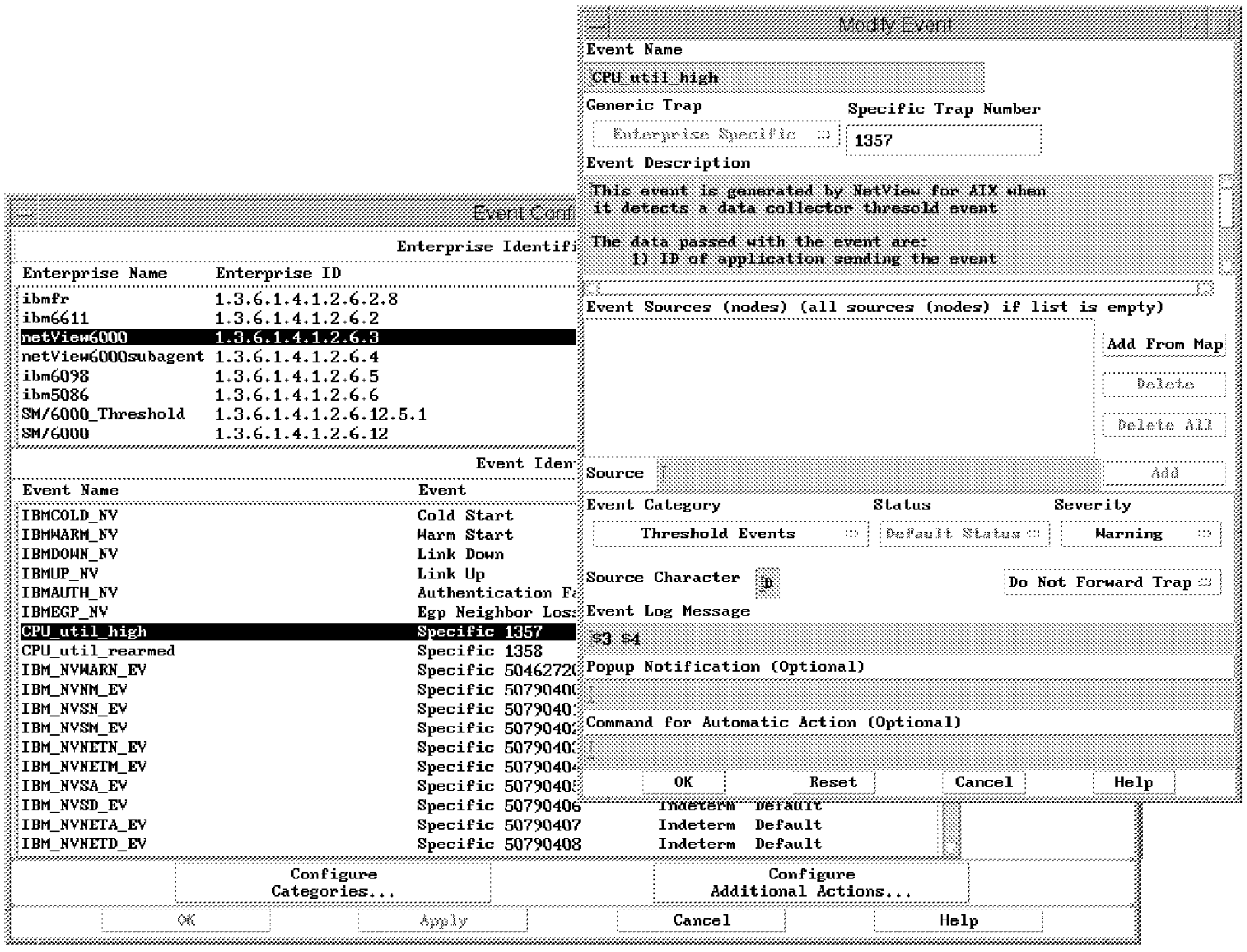
1. First we had to identify the value to monitor. The trapgend MIB reports CPU utilization as a percentage multiplied by 100. We therefore defined a *MIB Expression* that will return the true percentage figure. To do this we added the following lines to /usr/OV/conf/mibExpr.conf:

```
Trapgend_CPU_util \
"% utilization, as reported by trapgend subagent, \
(Percentage CPU busy)" \
 .1.3.6.1.4.1.2.6.4.5.1. 100 /
```

2. We selected Tools→Data Collection and Thresholds and followed the dialog to add a collection for our MIB expression.

3. When specifying thresholds, we set a polling interval of one minute. We also specified a threshold value of 50 and a rearm value of 100. The reason for this strange rearm value is that we want to disable the rearm function, so that we see threshold exception events for every poll where the MIB value exceeds the threshold. We also elected to use our own trap numbers, in place of the standard ones (we chose specific trap 1357 for the threshold, which implies 1358 for the rearm).

4. We configured the two new traps using the option Options→Event Configuration→Trap Customization from the menu bar (see Figure 73).



Event Name: CPU_util_high

Generic Trap: Enterprise Specific

Specific Trap Number: 1357

Event Description:

```
This event is generated by NetView for AIX when
 it detects a data collector thresold event

The data passed with the event are:
     1) ID of application sending the event
```

Event Sources (nodes) (all sources (nodes) if list is empty)

Event Category: Threshold Events

Status: Default Status

Severity: Warning

Source Character: D

Do Not Forward Trap

Event Log Message: $3 $4

Popup Notification (Optional)

Command for Automatic Action (Optional)

Enterprise Identifi

```
Enterprise Name      Enterprise ID
ibmfr                1.3.6.1.4.1.2.6.2.8
ibm6611              1.3.6.1.4.1.2.6.2
netView6000          1.3.6.1.4.1.2.6.3
netView6000subagent  1.3.6.1.4.1.2.6.4
ibm6098              1.3.6.1.4.1.2.6.5
ibm5086              1.3.6.1.4.1.2.6.6
SM/6000_Threshold    1.3.6.1.4.1.2.6.12.5.1
SM/6000              1.3.6.1.4.1.2.6.12
```

Event Iden

```
Event Name                              Event
IBMCOLD_NV                              Cold Start
IBMWARM_NV                              Warm Start
IBMDOWN_NV                              Link Down
IBMUP_NV                                Link Up
IBMAUTH_NV                              Authentication Fa
IBMEGP_NV                               Egp Neighbor Los
CPU_util_high                           Specific 1357
CPU_util_rearmed                        Specific 1358
IBM_NVWARN_EV                           Specific 5046272
IBM_NVNM_EV                             Specific 5079040
IBM_NVSN_EV                             Specific 5079040
IBM_NVSM_EV                             Specific 5079040
IBM_NVNETN_EV                           Specific 5079040
IBM_NVNETM_EV                           Specific 5079040
IBM_NVSA_EV                             Specific 5079040
IBM_NVSD_EV                             Specific 50790406        Indeterm  Default
IBM_NVNETA_EV                           Specific 50790407        Indeterm  Default
IBM_NVNETD_EV                           Specific 50790408        Indeterm  Default
```

Figure 73. Configuring the Threshold Event. The rearm event was configured in a similar way, except that we set the event category to Log Only (we will get a rearm event with every threshold event, so we are not interested in seeing them).

6.2.2.2 Constructing the Ruleset

We created the ruleset shown in Figure 72 on page 103 as follows:

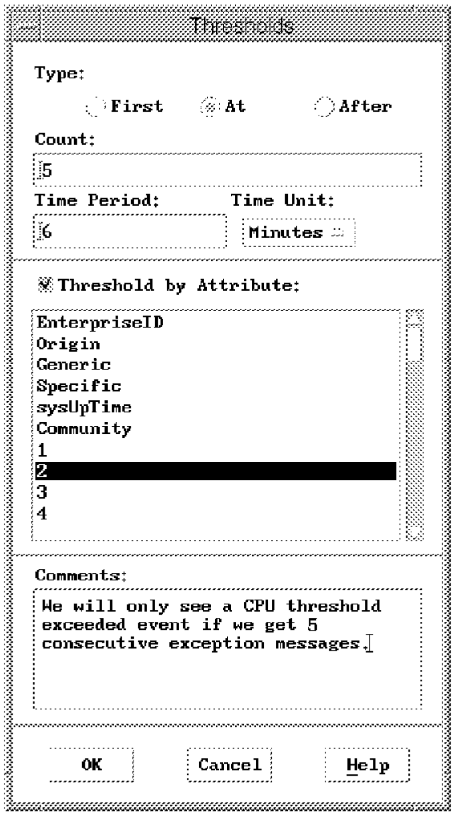
1. As before, the first step is to define the default behavior of the ruleset, by double-clicking the event source node. In this case we *only* want to see threshold events that pass the filter rule, so we set the default behavior to Block (see Figure 59 on page 89).

2. Our next step is to identify the CPU Load event. This is accomplished by dragging the Trap Settings node into the work area and selecting the following settings in the accompanying dialog box:

Enterprise Name	netview6000
Event Name	CPU_util_high (this is the name we chose when we defined the event)
Specific	Specific 1357

   This node should now be attached to the Event Stream node.

3. Next we can add the Thresholds node to the work area. Figure 74 shows the Threshold dialog and the settings we used in this example. Note that we set Type to At and Count to 5 because we want the ruleset to count five traps and then forward the fifth one only if all the traps occur within a six minute time period. Note also that we are checking that the second variable in the trap is the same. This is the host name or IP address field (see Figure 63 on page 95 for an explanation of this). If we did not consider the node name, we would see every fifth threshold event, no matter which node they came from.



Type: First [✓ At] After

Count: 5

Time Period: 6

Time Unit: Minutes

[✓] Threshold by Attribute:

EnterpriseID
Origin
Generic
Specific
sysUpTime
Community
1
[✓ 2]
3
4

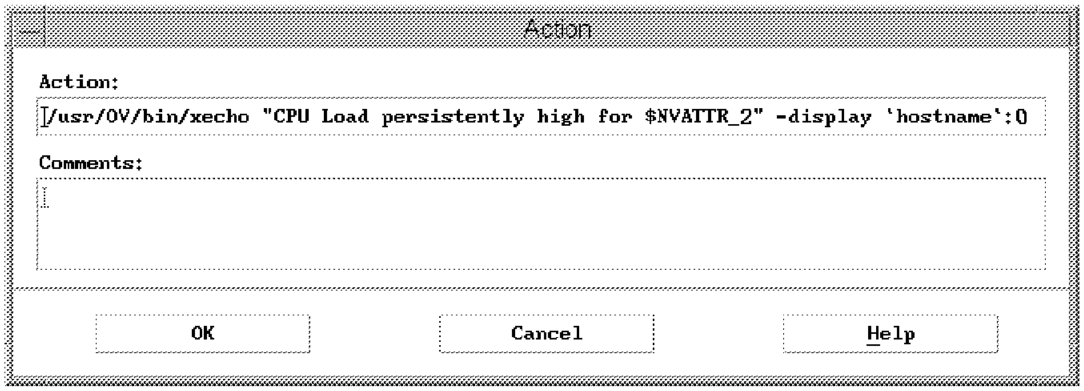
Comments:

We will only see a CPU threshold exceeded event if we get 5 consecutive exception messages.

Figure 74. Adding the Thresholds Node

4. Next we attach a Forward node to the Thresholds node. This node will act to display the event in the event display if it has been passed from the Threshold node.

5. In order to emphasize that this trap has been sent, we will also attach an Action node to the thresholds node. This node will act to display a message window on the screen. After dragging the Action node to the screen, we see a dialog box like that shown in Figure 75.



Action:

```
/usr/OV/bin/xecho "CPU Load persistently high for $NVATTR_2" -display `hostname`:0
```

Comments:

Figure 75. Adding the Action Node

Notice that in the command to be executed we are using the environment variable, NVATTR_2, which will contain the second variable from the trap. In this case (as is the case for all NetView for AIX internal traps) this is the node name of the machine that the trap is about. You can get a list of all the environment variables available by selecting **Help** in the Action node configuration dialog box. Using the `hostname`:0 construction will cause the xecho pop-up message to appear on the hft screen of the machine running NetView for AIX.

Running Display Commands in Action Nodes

If you want to run an automated command that uses X-Windows display functions, you need to specify the display environment, as in the example shown here. The reason for this is that commands specified in the Action and Inline Action nodes execute under a daemon (actionsvr and nvcorrd respectively). They therefore do not inherit the environment variables of the application that invoked the ruleset.

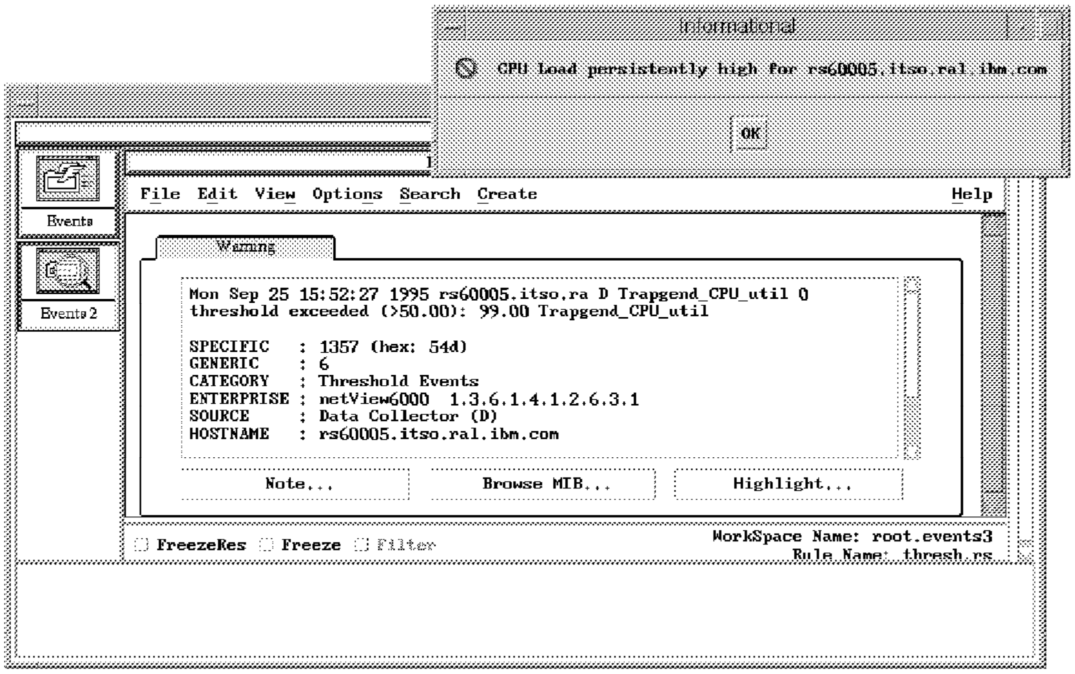
6.2.2.3 Testing the Ruleset

Now we can test our ruleset by completing the following steps:

1. Start the threshold polling by selecting Tools→Data Collection and Thresholds and clicking on **Resume** and then **OK**.

2. Run a looping command on one of the machines being polled to drive up CPU utilization. We used the `lslpp` command within a WHILE loop for this.

After five minutes or so, you should receive an event and see a message displayed on your screen like that shown in Figure 76 on page 107.



Informational
OK
Events
Events 2
File Edit View Options Search Create Help
Warning
Note... Browse MIB... Highlight...
FreezeRes Freeze Filter

```
CPU Load persistently high for rs60005.itso.ral.ibm.com
```

```
Mon Sep 25 15:52:27 1995 rs60005.itso.ra D Trapgend_CPU_util 0
threshold exceeded (>50.00): 99.00 Trapgend_CPU_util

SPECIFIC   : 1357 (hex: 54d)
GENERIC    : 6
CATEGORY   : Threshold Events
ENTERPRISE : netView6000  1.3.6.1.4.1.2.6.3.1
SOURCE     : Data Collector (D)
HOSTNAME   : rs60005.itso.ral.ibm.com
```

```
WorkSpace Name: root.events3
Rule Name: thresh.rs
```

Figure 76. Result of a CPU Load Trap Passing through the Threshold Ruleset

6.2.3 Using Thresholds in Combination with Correlation

In this example, we will combine the threshold function with the correlation function in order to define a more specialized ruleset. This example will use the last example as its starting point. However, we will add two important sets of contingencies.

1. In addition to seeing every fifth CPU load event, we will run a second threshold monitor for IP interface utilization, and set up a ruleset to show an event if this monitor generates five events within six minutes.
2. If an interface busy event follows a CPU load event within one minute, we will display the CPU load event regardless of the threshold settings.

The idea is that bursts of high CPU utilization and high network I/O are acceptable, so long as they do not persist and do not occur together.

Figure 77 displays a flow of what we want to obtain with this ruleset.

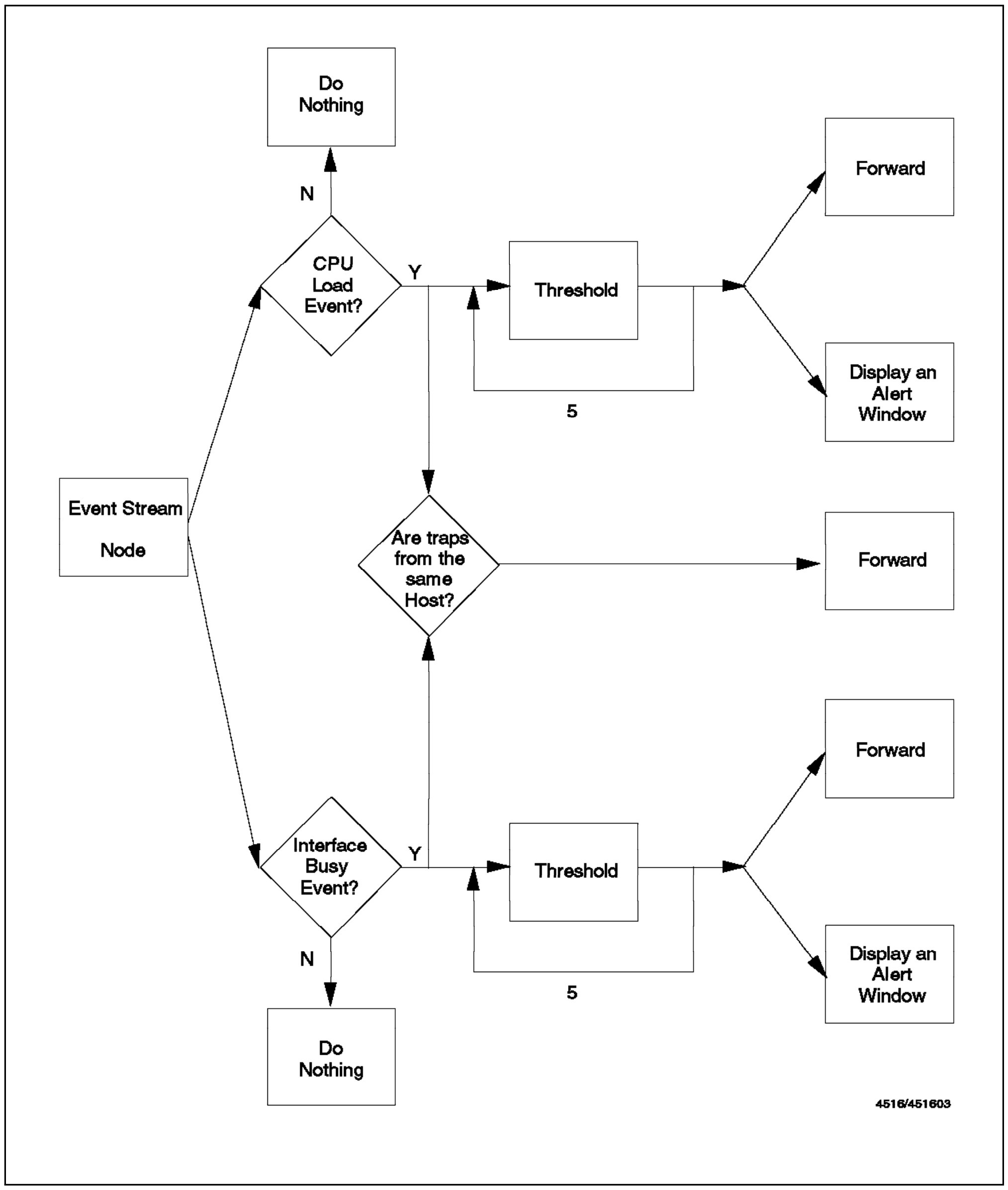


Figure 77. Flowchart of Ruleset that Combines Thresholds with Correlation

The ruleset that we constructed for this example is displayed in Figure 78 on page 109.

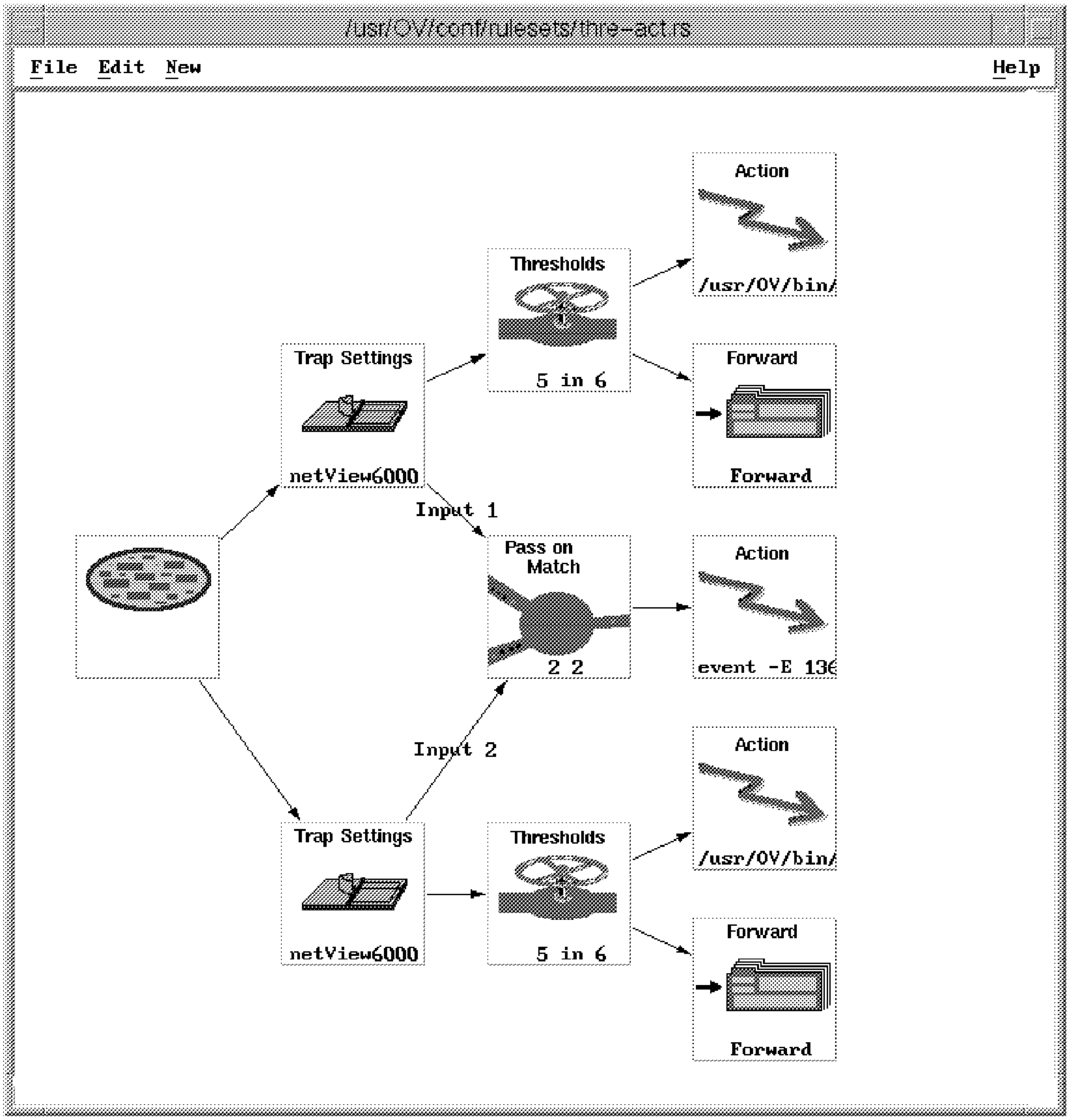


Figure 78. Ruleset that Combines Thresholds with Correlation

6.2.3.1 Setting up the Threshold Monitor

We followed the same procedure as for the CPU utilization monitor (see 6.2.2, "Suppressing Events by Setting Thresholds" on page 101) except that the MIB variable we were monitoring was IfInOctets (.1.3.6.1.2.1.2.2.1.10). We again used a one minute polling period with a threshold value this time of 100,000 (that is, we designated anything over one hundred thousand bytes per minute as a high interface utilization).

We also created new threshold and rearm traps for this event using specific trap numbers 1359 and 1360.

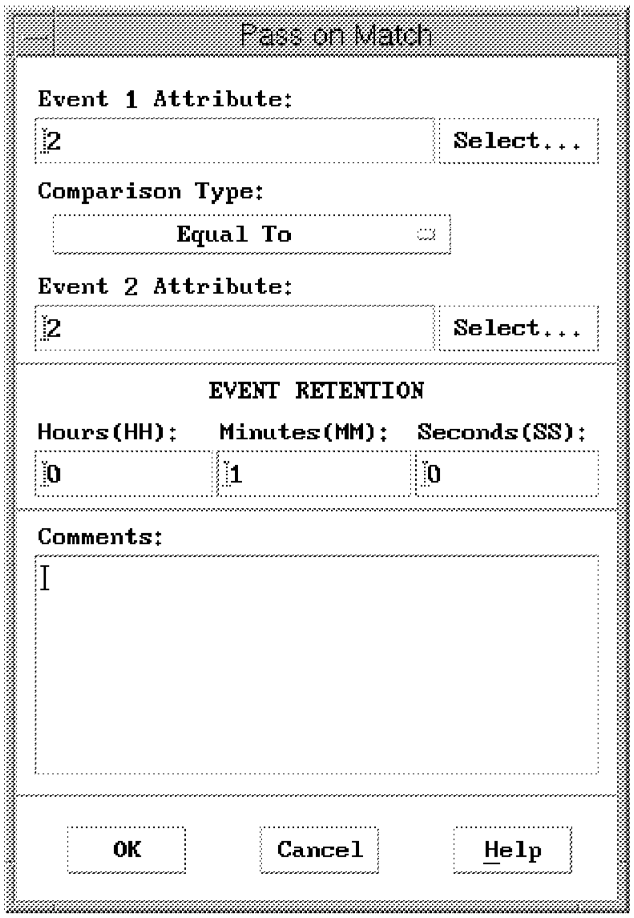
6.2.3.2 Construction the Ruleset

Use the following sequence of actions to create this ruleset:

1. First, click on File→Save As to save the thresh.rs ruleset from the last example with a new name.
2. Next, to create a second set of nodes exactly like the existing ones, use File→Include and select thresh.rs again. The Include function allows you to combine two rulesets together, so you will now have a ruleset with the thresh.rs nodes duplicated. You then need to modify the second Trap Settings node. Double-click it and change the settings as follows:

Enterprise Name	netview6000
Event Name	IP_octets_high
Specific	Specific 1359

3. Now we can add the Pass on Match node to the work area so that it appears as it does in Figure 78 on page 109. The values we put into the Pass on Match dialog box are shown in Figure 79.


Pass on Match

Event 1 Attribute: 2

Comparison Type: Equal To

Event 2 Attribute: 2

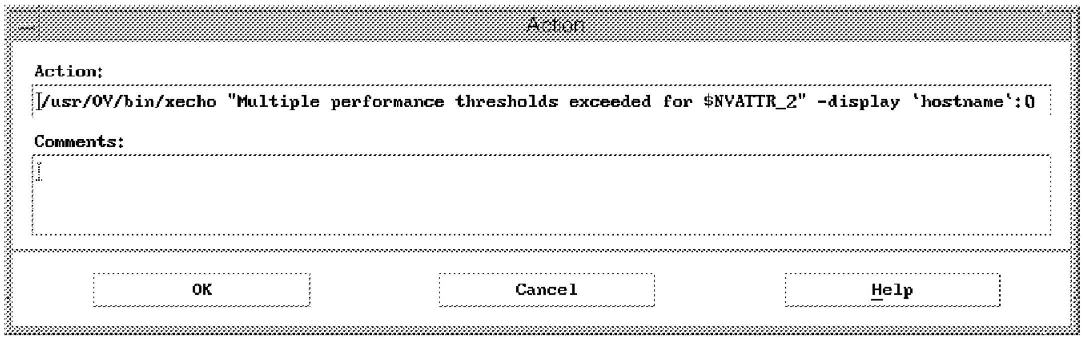
EVENT RETENTION

Hours(HH): 0 Minutes(MM): 1 Seconds(SS): 0

Comments:

Figure 79. Configuring the Pass on Match Dialog

4. The final Action node (see Figure 78 on page 109) generates a popup window that warns that two thresholds have been exceeded together. Figure 80 on page 111 shows the command we entered for this.



```
/usr/OV/bin/xecho "Multiple performance thresholds exceeded for $NVATTR_2" -display `hostname`:0
```

Figure 80. Configuring the Action Node for Combined Thresholds

5. Next we can test the configuration by creating a dynamic events display with our new ruleset, starting the threshold data collection and then creating both high CPU and high interface utilization. An easier way to test the ruleset is to simulate the threshold events using the following commands:

```
event -h 'hostname' -E 1357        (for the CPU trap)

event -h 'hostname' -E 1359        (for the interface utilization trap)
```

   We will only get the fifth CPU load trap or the fifth interface utilization within a minute unless an interface utilization trap follows a CPU load trap within a one minute period. In this case a pop-up warning will be displayed.

6.2.4 Automated Paging and E-Mail Notifications

In the last two examples we showed how you can use the ruleset Action function to display a message window that informs you about receiving certain traps. While this has some merit, message windows do not have much value if you are away from the workstation. Fortunately, NetView for AIX V4 provides the capability to inform you about important events wherever you are. This, combined with the logic processing in the ruleset nodes, is a powerful function because you can automate notification on the basis of very specific criteria. In our lab, we set up a ruleset that notified us about a particular router going down by initiating a call to our pager and sending e-mail. Figure 81 shows the flowchart of our automated notification ruleset. Figure 82 on page 113 shows the completed ruleset.

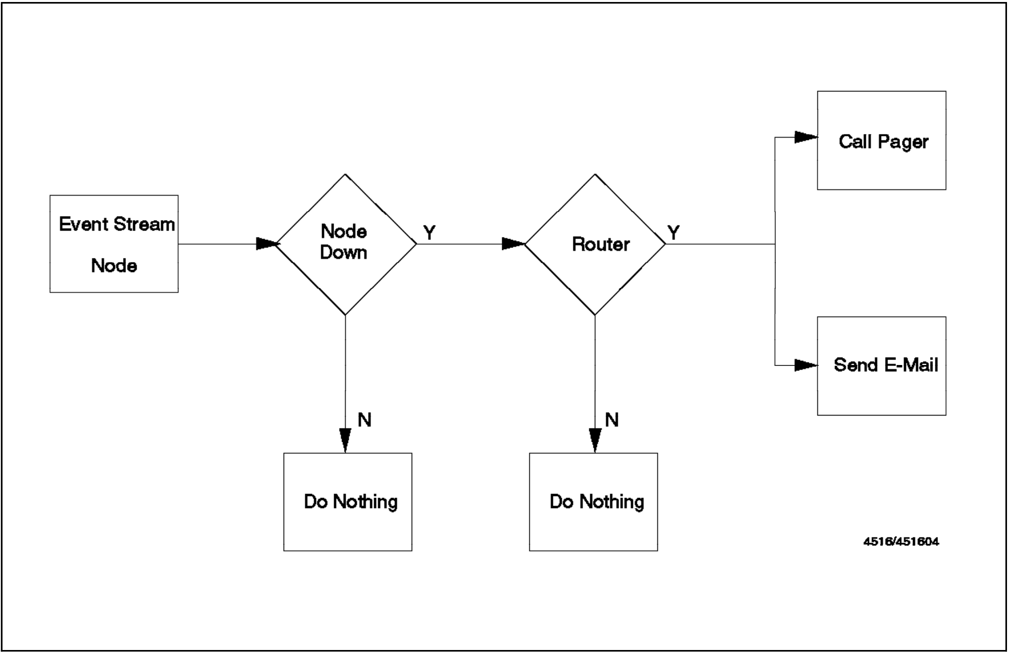


Figure 81. Flowchart of Automated Notification Example

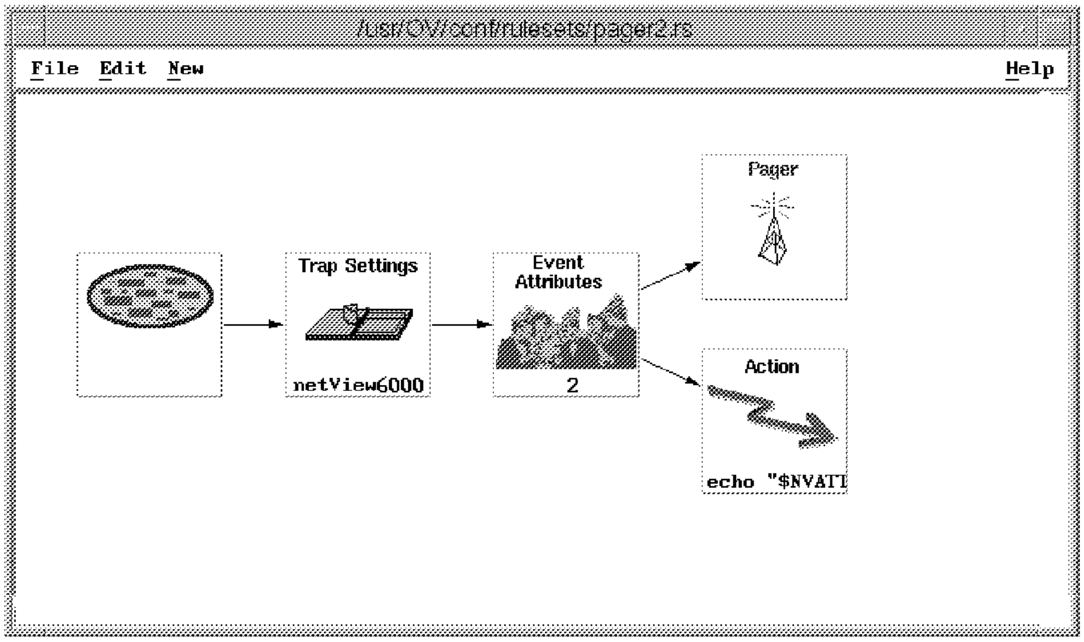


Figure 82. Complete Ruleset of Automated Notification Example

6.2.4.1 Constructing the Ruleset

We created this ruleset using the following sequence of actions:

1. Our first step was to connect a modem. In our lab, we connected an IBM 5853 modem to our RISC machine and defined a TTY. The TTY was defined via SMIT by progressing through the following menu options:

   a. Devices

   b. TTY

   c. Add a TTY

   d. tty rs232 Asynchronous Terminal

   e. sa1 Available 00-00-S2 Standard I/O Serial Port 2

   The settings for our TTY are shown in Figure 83.

```
TTY                                       tty1
TTY type                                  tty
TTY interface                             rs232
Description                               Asynchronous Terminal
Status                                    Available
Location                                  00-00-S2-00
Parent adapter                            sa1
PORT number                               s2
BAUD rate                                 2400
PARITY                                    none
BITS per character                        8
Number of STOP BITS                       1
XON-XOFF handshaking                      yes
RTS-CTS handshaking                       no
CODESET map file                          sbcs
STATE to be configured at boot time       available
Read Trigger                              3
Transmit Buffer Count                     16
```

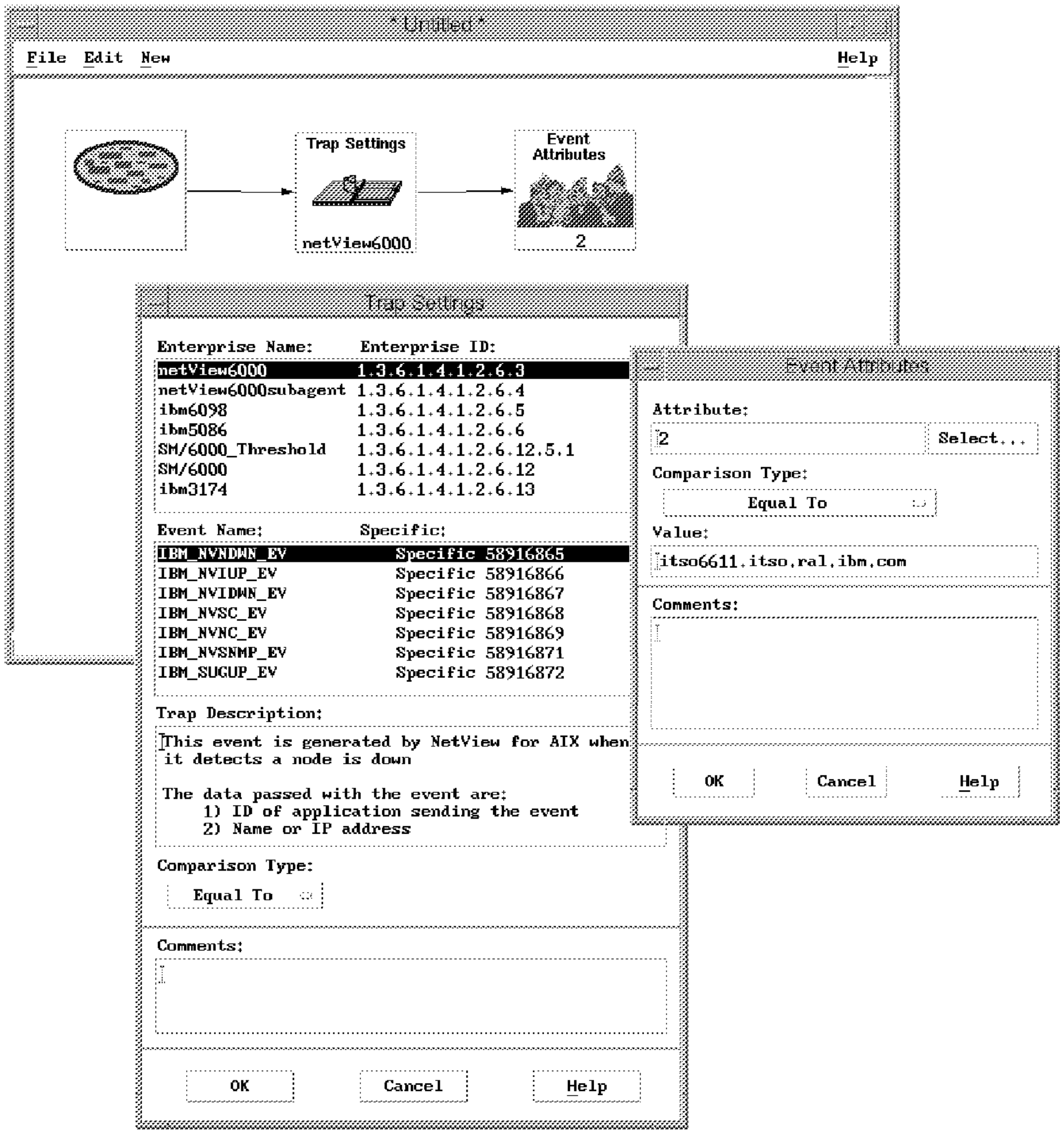
Figure 83. Our TTY Settings

2. NetView for AIX provides configuration files that work with pagers and modems. For our purposes, we needed to edit /usr/OV/conf/nvpager.config so that the paging program could work with our modem. Figure 84 on page 114 shows a listing of our file. Although we did not have to modify it, you may want to note that another file, /usr/OV/conf/nv.carriers, contains information about each of the defined pager carriers. If the company that provides your paging service is not listed you may have to modify this file to add it.

```
#
# GSC Paging configuration file
#
# Outside Line # (p=Pause): 9p
#    This is any dialing that needs to be done for the attached modem to
#    get an outside dial tone.  Numeric pager numbers as well as carrier
#    phone number should NOT include this.  This will be dialed each time
#    a call is made by the modem.
#    Valid characters: 0123456789p
Outside Line # (p=Pause): 9p

# Default Modem File: newhayes.modem
#    This is the modem information file that describes the initialization
#    needed for the attached modem and other control information specific
#    to each type of modem.
Default Modem File: ibm5853.modem

# Default Baud Rate (300,1200,2400,4800,9600,19200,38400): 300
#    This is the baud rate between the computer and modem when direct-dial
#    numeric pagers are contacted and when the carrier′s direct-dial DTMF
#    number is used for numeric pagers.
#    Valid values: 300,1200,2400,4800,9600,19200,38400
Default Baud Rate (300,1200,2400,4800,9600,19200,38400): 300

# Default Data Bits (7,8): 7
#    This is the number of data bits between the computer and modem when
#    direct-dial numeric pagers are contacted and when the carrier′s direct-
#    dial DTMF number is used for numeric pagers.
#    Valid values: 7,8
Default Data Bits (7,8): 7

# Default Parity (N,E,O): E
#    This is the parity between the computer and modem when direct-dial numeric
#    pagers are contacted and when the carrier′s direct-dial DTMF number is
#    used for numeric pagers.
#    Valid values: N,E,O
Default Parity (N,E,O): E

# Default Stop Bits (1,2): 1
#    This is the number of stop bits between the computer and modem when
#    direct-dial numeric pagers are contacted and when the carrier′s direct-
#    dial DTMF number is used for numeric pagers.
#    Valid values: 1,2
Default Stop Bits (1,2): 1

Default Device: tty1
#    This is the device that the modem is attached to.  It must be in the /dev
#    directory.  If your modem is attached to another tty port, this must be
#    changed.
# Default Device: tty0
```

Figure 84. Our /usr/OV/conf/nvpager.config File

3. Our first step was to identify exactly the circumstances under which the notification was to take place. For this example, this entailed adding both a Trap Setting node and an Event Attribute node so that we could identify that it was both a Node Down trap and that the trap was reporting on a particular router in our lab. Figure 85 on page 115 shows how we identified the trap.



Enterprise Name: Enterprise ID:

```
netView6000            1.3.6.1.4.1.2.6.3
netView6000subagent    1.3.6.1.4.1.2.6.4
ibm6098                1.3.6.1.4.1.2.6.5
ibm5086                1.3.6.1.4.1.2.6.6
SM/6000_Threshold      1.3.6.1.4.1.2.6.12.5.1
SM/6000                1.3.6.1.4.1.2.6.12
ibm3174                1.3.6.1.4.1.2.6.13
```

Event Name: Specific:

```
IBM_NVNDWN_EV          Specific 58916865
IBM_NVIUP_EV           Specific 58916866
IBM_NVIDWN_EV          Specific 58916867
IBM_NVSC_EV            Specific 58916868
IBM_NVNC_EV            Specific 58916869
IBM_NVSNMP_EV          Specific 58916871
IBM_SUGUP_EV           Specific 58916872
```

Trap Description:

```
This event is generated by NetView for AIX when
it detects a node is down

The data passed with the event are:
    1) ID of application sending the event
    2) Name or IP address
```

Comparison Type: Equal To

Comments:

Attribute: 2

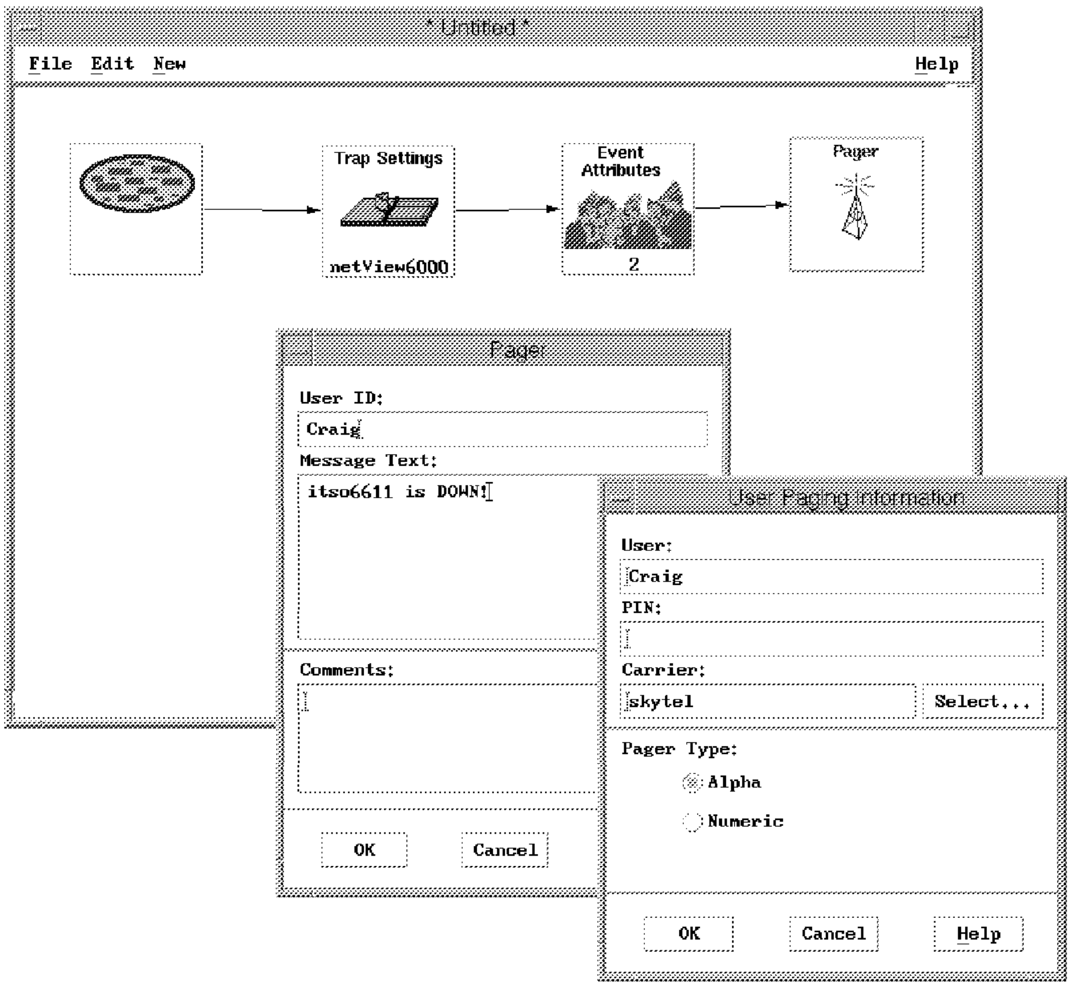
Comparison Type: Equal To

Value: itso6611.itso.ral.ibm.com

Comments:

Figure 85. Identifying a Node Down Trap for a Particular Router

4. Our next step was to connect a Pager node to the Events Attributes node. In the pager dialog, the User ID field refers to IDs that are defined in NetView for AIX′s security system. Since we had not defined one yet, we just typed a name in the field. When we clicked on **OK**, NetView determined that this name had not been defined and put up a dialog called User Paging Information. In this dialog, we typed in our PIN and chose our carrier from the defined list. After selecting **OK** a file called /usr/OV/security/C/Users/Craig was created. Figure 85 shows how the paging information was configured.



Pager

User ID:
Craig

Message Text:
itso6611 is DOWN!

Comments:

User Paging Information

User:
Craig

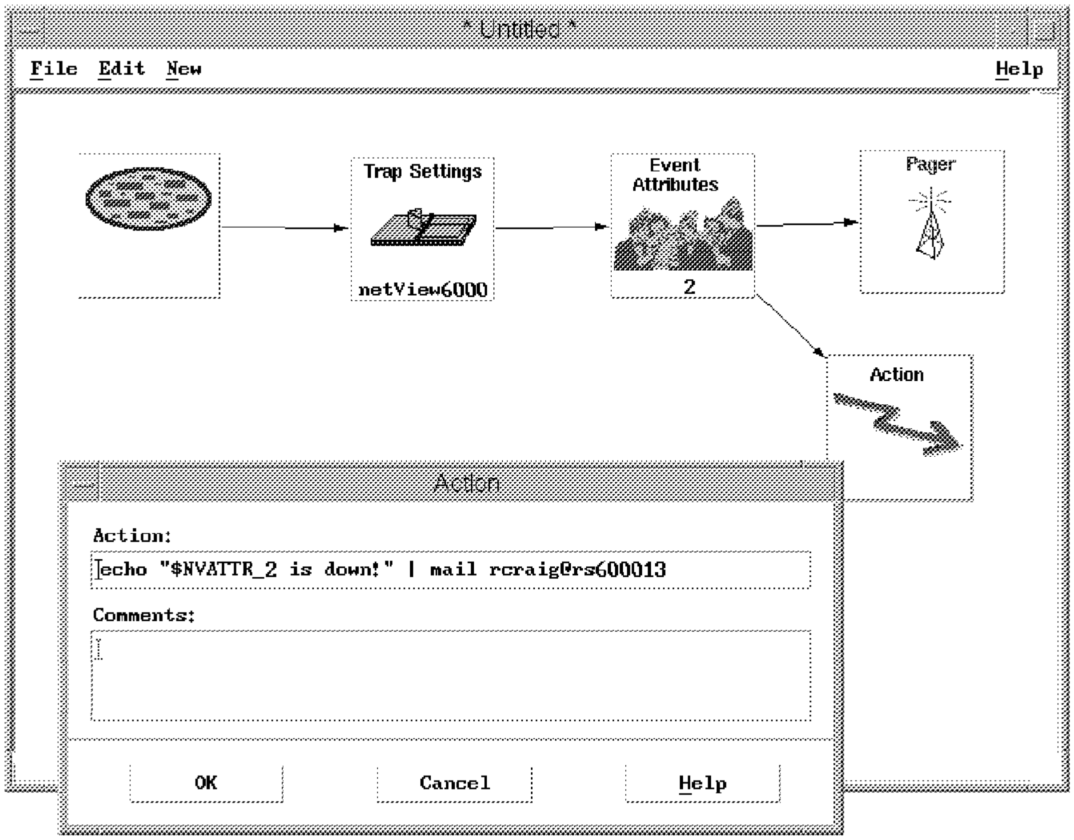
PIN:

Carrier:
skytel

Pager Type:
[✓ Alpha]
Numeric

Figure 86. Configuring Pager Dialogs

5. Our next step was to add an action that sent e-mail. For us, this was a simple Action node invoking the mail command. Figure 87 on page 117 shows how we added this command.



```
echo "$NVATTR_2 is down!" | mail rcraig@rs600013
```

Figure 87. Adding Action for Sending Automatic E-Mail

6.2.4.2 Testing the Ruleset

We could have created a dynamic workspace to test this ruleset, in the same way as in the previous examples. However, in this case the ruleset is simply performing an automation function, so there is no need to involve an operator display. We therefore want this ruleset to be registered to run in the background, instead of being invoked by an nvevents workspace.

Follow these steps to register a ruleset for background processing:

- Place the filename of the ruleset in file /usr/OV/conf/ESE.automation (you can register additional rulesets by adding multiple lines).
- Stop and restart the actionsvr daemon, using the `ovstart` and `ovstop` commands.

To test the ruleset we used the `event` command as before (see 6.2.1.2, “Testing the Ruleset” on page 98) specifying itso6611 as the node that was down. Within a few seconds, our pager bleeped and showed the error message and we received the e-mail message.

6.2.5 Using Traps to Override Status Color and Severity

Rulesets makes it easy to change the color-coded status of nodes on a submap and also to change the severity of traps. This can be accomplished by using the Override action node. In this example we show how to change the status color for a node on the basis of threshold traps from System Monitor agents. We also changed the severity of events based upon the type of device on which the trap is reporting.

Specifically, we made use of a threshold that was set up to monitor the root file system on a RISC machine. If the amount of space used exceeded 85%, a trap was forwarded to an event workspace and the color of its symbol on the map changed to purple (this is one of two user-defined statuses that NetView provides). If the used space went below 85%, a rearm trap was forwarded and the symbol changed back to green. Also, if a node down trap was received and reported on a router, we changed the severity of the event to Major. Figure 88 displays a flowchart of what our ruleset accomplished.

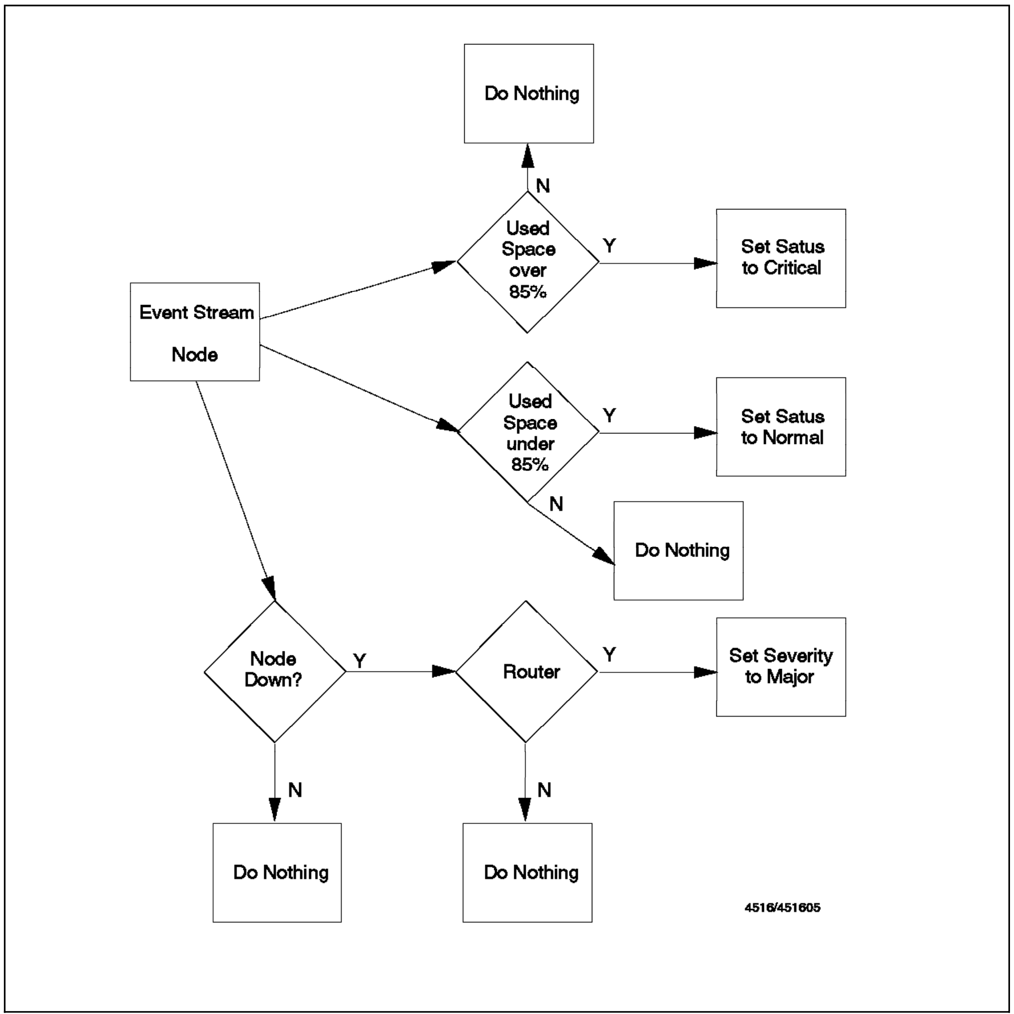


Figure 88. Flowchart of Override Example

Figure 89 on page 119 displays the completed ruleset that was used in this example.

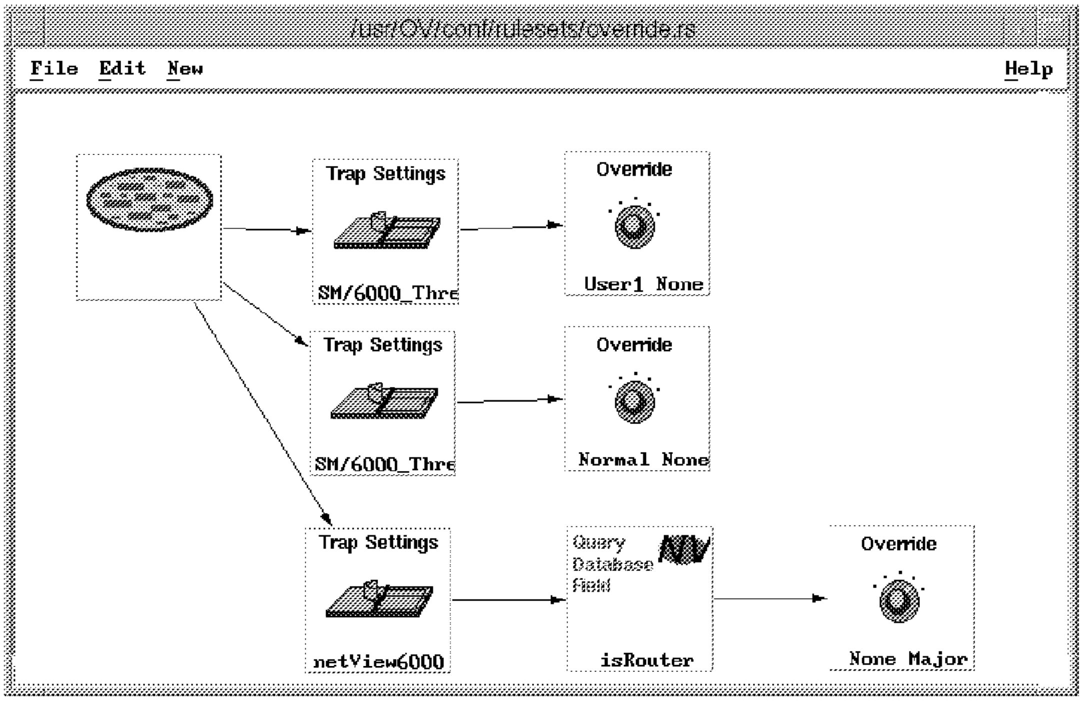
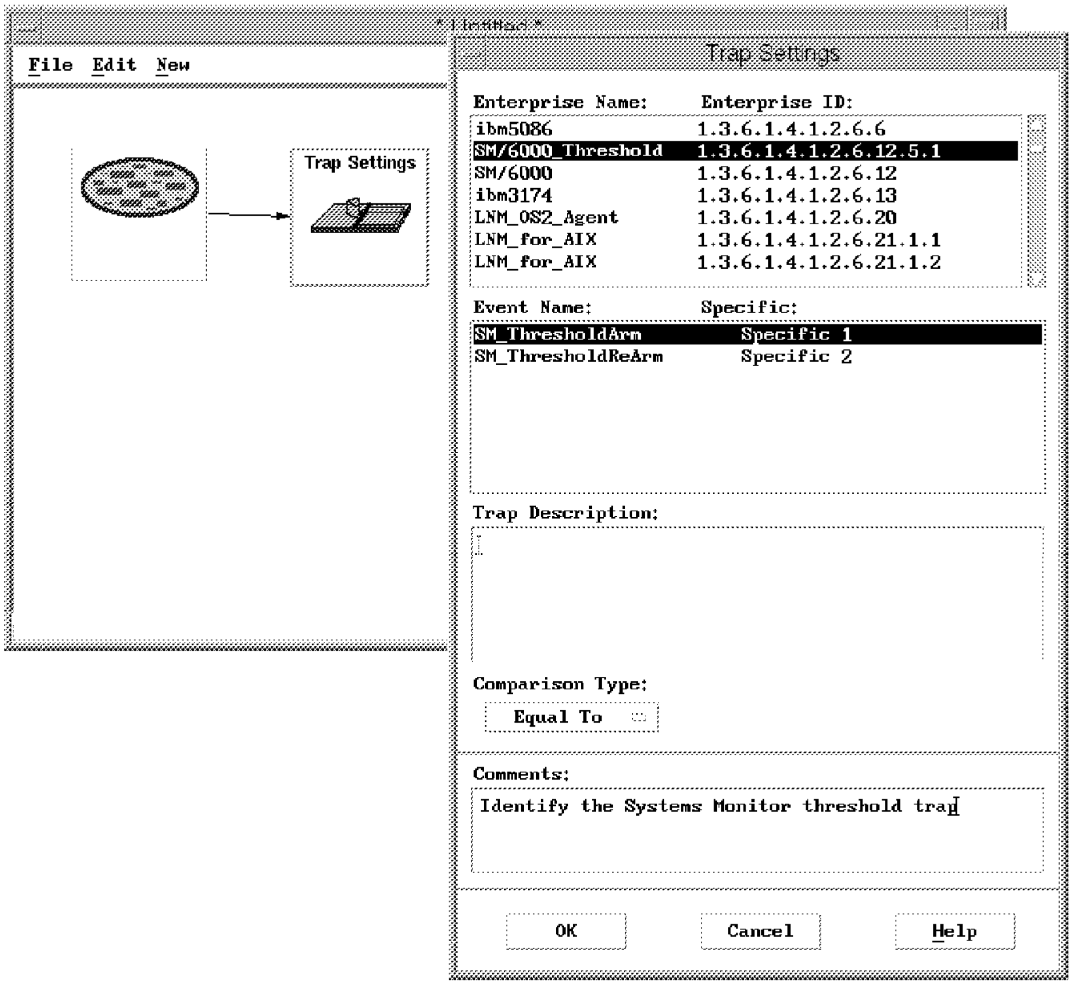


Figure 89. Ruleset that Overrides Status and Severity

6.2.5.1 Constructing the Ruleset

The following steps describe how this rule was constructed.

1. First, we identified the traps by using the Trap Settings function. In the upper flow, we identified the Systems Monitor ThresholdArm trap as shown in Figure 90 on page 120. In the middle flow we identified the Systems Monitor ThresholdReArm trap. Finally, in the lower flow, we identified the node down trap.



Trap Settings

Enterprise Name: Enterprise ID:

```
ibm5086                1.3.6.1.4.1.2.6.6
SM/6000_Threshold      1.3.6.1.4.1.2.6.12.5.1
SM/6000                1.3.6.1.4.1.2.6.12
ibm3174                1.3.6.1.4.1.2.6.13
LNM_OS2_Agent          1.3.6.1.4.1.2.6.20
LNM_for_AIX            1.3.6.1.4.1.2.6.21.1.1
LNM_for_AIX            1.3.6.1.4.1.2.6.21.1.2
```

Event Name: Specific:

```
SM_ThresholdArm          Specific 1
SM_ThresholdReArm        Specific 2
```

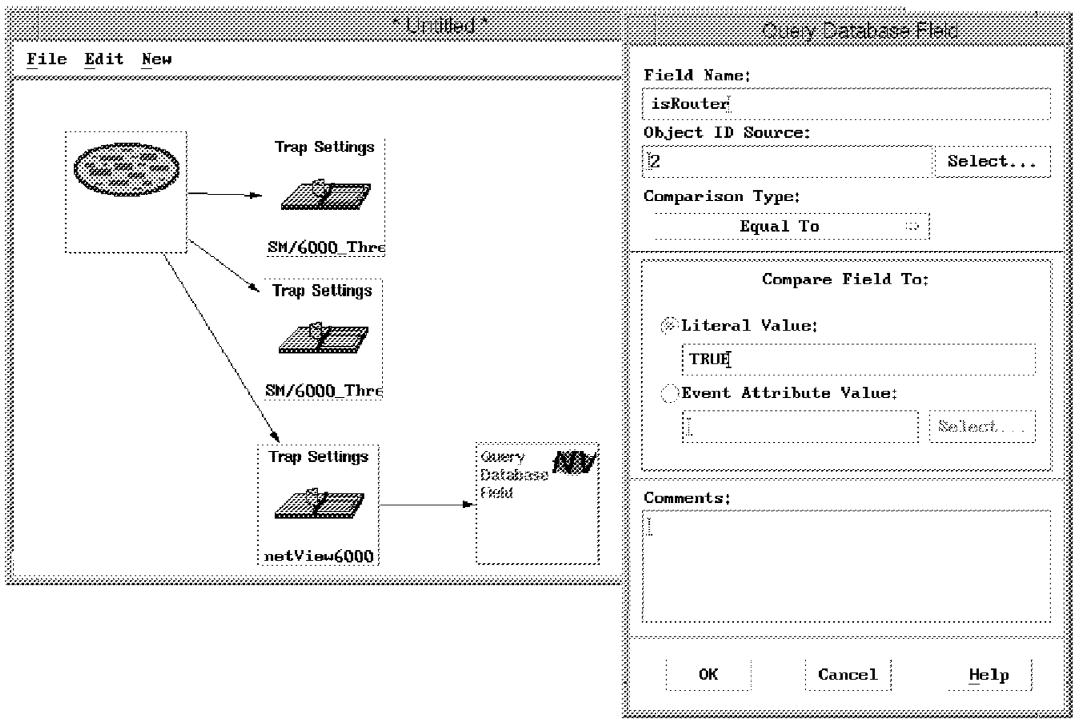
Trap Description:

Comparison Type: Equal To

Comments: Identify the Systems Monitor threshold trap

Figure 90. Identifying The Systems Monitor ThresholdArm Trap

2. To the Trap Setting node for Node Down traps, we attached a Query Database Field node. The database field we checked was isRouter which is set to True if the node in question is a router. In this way we could distinguish router nodes from other, less important, nodes. Figure 91 on page 121 shows how we configured this node. Notice that the Object ID Source field is set to 2 because Node Down is a NetView event, as we described in 6.2.1, “Clearing Outstanding Events via Correlation” on page 92.



Field Name:
isRouter
Object ID Source:
2
Select...
Comparison Type:
Equal To
Compare Field To:
(●) Literal Value:
TRUE
( ) Event Attribute Value:
Select...
Comments:
OK Cancel Help

Figure 91. Configuring the Query Database Field Dialog

3. To each flow, we then attached an Override node. It is with this node that controls changes in color or severity. In the upper flow we changed the status to User2. This means that a threshold trap will result in the node status changing to User2 which causes the color of the symbol on the submap to turn purple. In the middle flow, the status was changed to Normal which means that the symbol will turn green when a Rearm trap is received. In the lower flow, the severity was changed to Major which means that node down events from routers will have a major severity setting instead of the default (Indeterminate) severity. This does not affect the color of the symbol on the submap (which will change to red as a result of Node Down anyway) but it does change the color of the event card tab to red. Figure 92 on page 122 displays the settings of the Override Nodes.

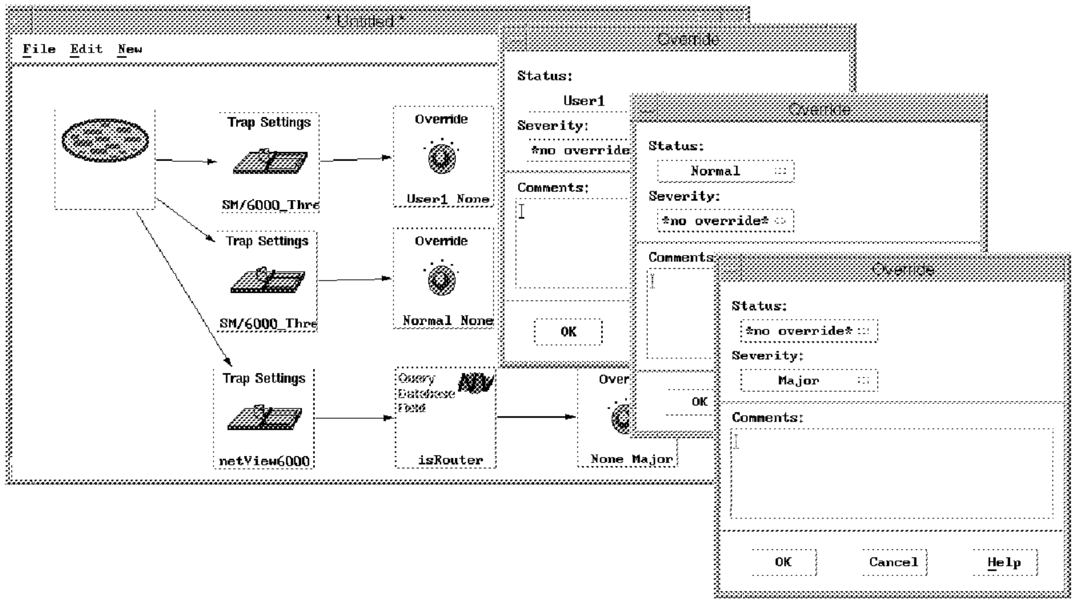
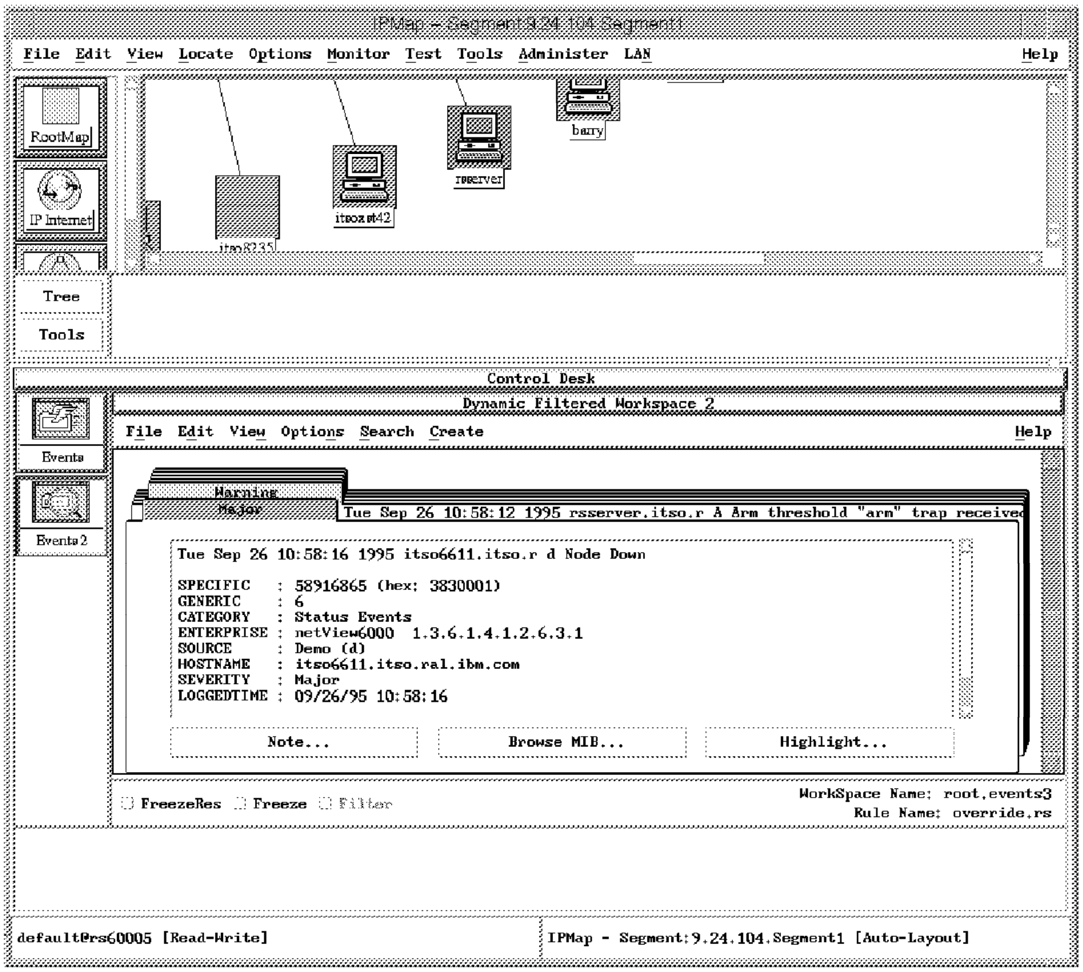


Figure 92. Setting the Override Function to Change Status and Severity

6.2.5.2 Testing the Ruleset

Since this ruleset works with different types of traps, we had to use more than one method to generate them. To test the part of the ruleset that overrides event severity we created simulated traps using the event command, as in the case of several of the previous examples (see 6.2.1.2, "Testing the Ruleset" on page 98). For the Systems Monitor events we could not easily simulate them, so we set up a Threshold Table entry on a machine running the Systems Monitor Mid-Level Manager, and then filled up a file system on the monitored machine to trigger the threshold trap. We will not go into the detail of how to do this here. Refer to *IBM Systems Monitor: Anatomy of a Smart Agent* (GG24-4398) for a full description of the features of Systems Monitor and configuration examples.

We tested the ruleset by selecting it in a new dynamic workspace. The result of the arrival of the Systems Monitor threshold event and a Node Down event for a router is shown in Figure 93 on page 123.


IPMap - Segment:9.24.104.Segment1
File Edit View Locate Options Monitor Test Tools Administer LAN Help
RootMap
IP Internet
barry
rsserver
itsozot42
itso8235
Tree
Tools
Control Desk
Dynamic Filtered Workspace 2
File Edit View Options Search Create Help
Events
Events2
Warning
Major
Note...
Browse MIB...
Highlight...
FreezeRes
Freeze
Filter

```
Tue Sep 26 10:58:12 1995 rsserver.itso.r A Arm threshold "arm" trap receive

Tue Sep 26 10:58:16 1995 itso6611.itso.r d Node Down

SPECIFIC   : 58916865 (hex: 3830001)
GENERIC    : 6
CATEGORY   : Status Events
ENTERPRISE : netView6000  1.3.6.1.4.1.2.6.3.1
SOURCE     : Demo (d)
HOSTNAME   : itso6611.itso.ral.ibm.com
SEVERITY   : Major
LOGGEDTIME : 09/26/95 10:58:16
```

```
WorkSpace Name: root,events3
     Rule Name: override.rs

default@rs60005 [Read-Write]        IPMap - Segment:9.24.104.Segment1 [Auto-Layout]
```

Figure 93. Results of Status and Severity Overrides

Note that in this case we could not run this ruleset in the background, as we did in 6.2.4, "Automated Paging and E-Mail Notifications" on page 112. The reason is that the Override node *only* operates if the ruleset forwards the node to be displayed by nvevents. The reason for this is that overriding of IP node status color is achieved by updating the map database. We have seen inFigure 1 on page 8 that the map database is updated by the end user interface processes. It is therefore not possible to update symbol status directly from the daemon where the rulesets are processed. This design provides some extra flexibility. For example, you may want to highlight an event by changing the status color of a node for one user but not for another user. If the two users have different NetView for AIX maps open they only need to run different rulesets to achieve this objective.

Changing Node Colors Without Displaying Events

There is one unfortunate side effect of the fact that the Override node only works for events that are forwarded to nvevents. What if you want to suppress the events from being displayed, but still update the status color of the affected node (there is an example of this in 7.1.4, "The wtdepend Sample Application" on page 187)? The best solution is to override both the status color *and* the event severity. By setting a severity of, for example Minor, you can then suppress the events from appearing by selecting not to display Minor events when you create the dynamic workspace.

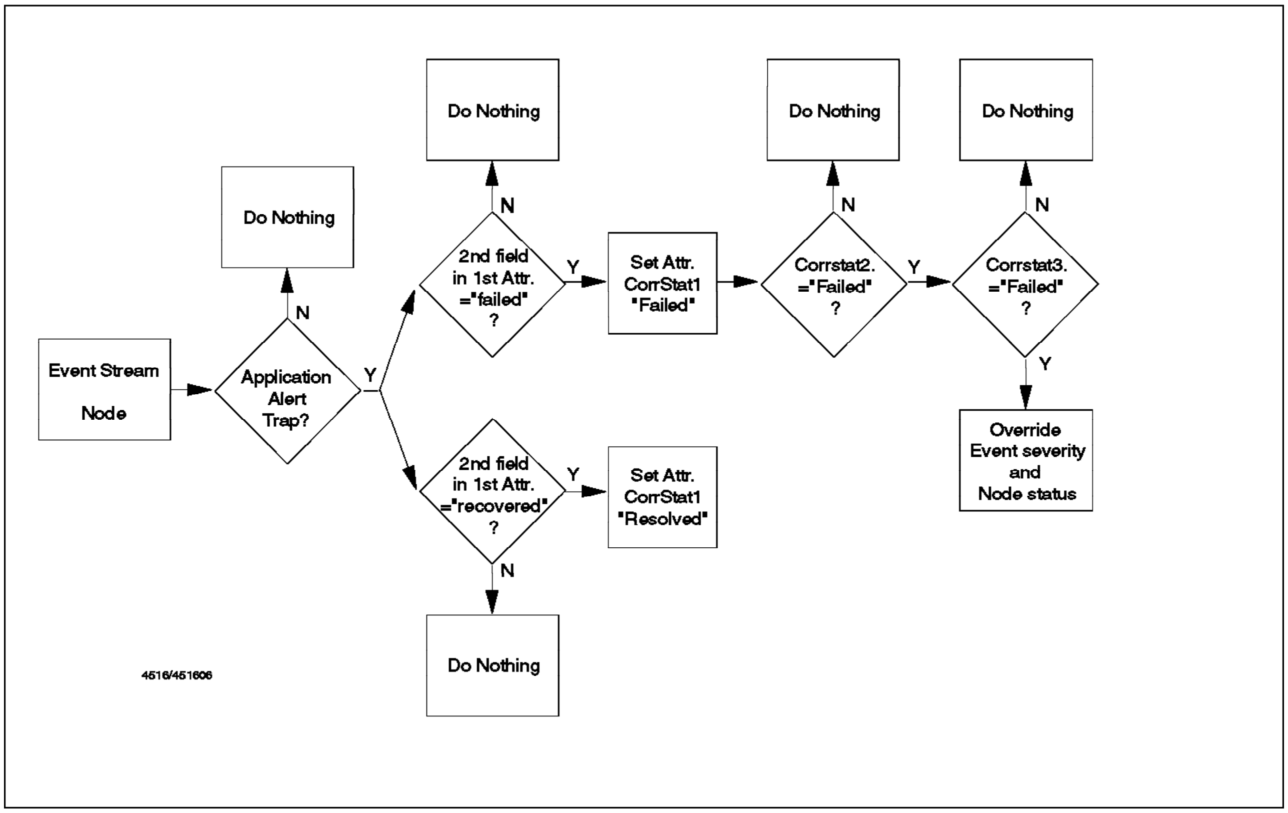
6.2.6 Setting Correlation States

In NetView for AIX, network objects are defined by a variety of different states and variables. These states and variables are useful for a wide variety of reasons and are often the point of integration for communication among applications. ESE provides several methods to change the value of these states and variables. In this example, we will learn about one such method called *Correlation States*.

In NetView for AIX Version 4, the ruleset function includes five new object database fields called corrstat1, corrstat2, corrstat3, corrstat4 and corrstat5. These fields are generic string fields that can be set on the basis of trap activity by using the Set State action node. The first time the Set State function is used, corrstat1 is set to a user-defined value. The next time the function is used, corrstat1 takes on the new value and passes the old value to corrstat2. This passage of values continues to the rest of the corrstat variables with each use of the set state function.

How would you use this function? Suppose you have some automatic process which goes through a series of recovery steps. You could use the corrstat fields to track the current state, plus the preceding states.

In this example, we imagine that we have an application that sends traps to tell the operator of problems it is encountering. The application has a built-in recovery function, so normally we are not concerned about the traps. However, if the automatic recovery fails three times in succession we will have to intervene manually. We will show how to use the corrstat fields to track this process. Figure 94 on page 125 shows a flow of what we want this ruleset to do.

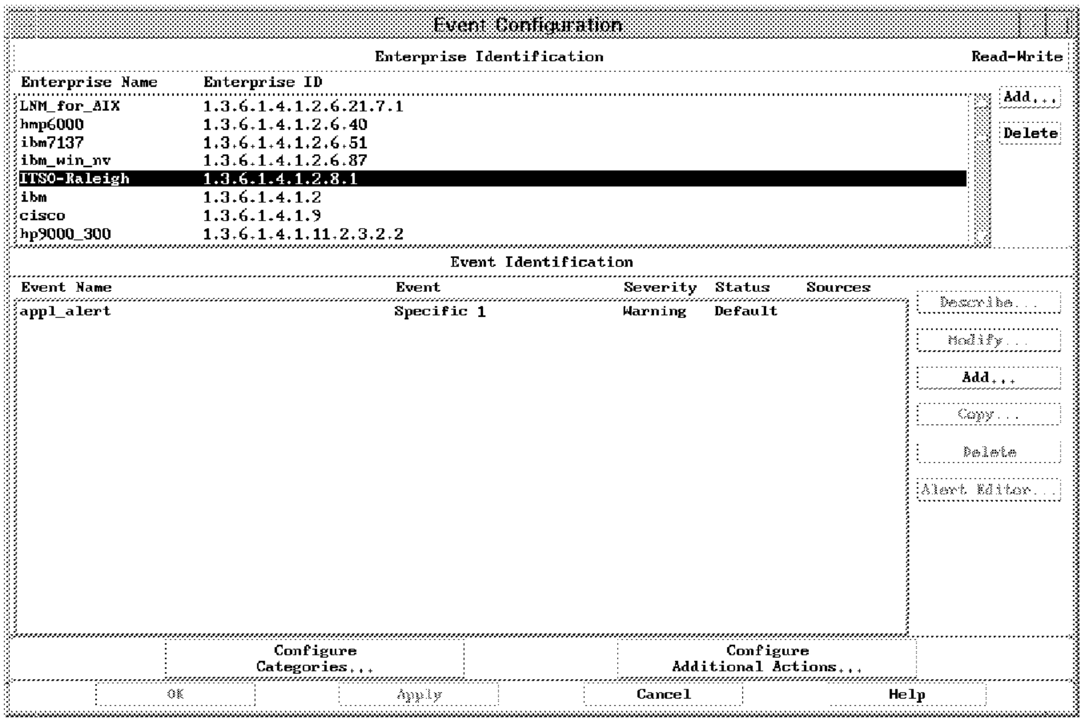


Event Stream Node
Application Alert Trap?
Y
N
Do Nothing
2nd field in 1st Attr. ="failed" ?
Y
N
Do Nothing
Set Attr. CorrStat1 "Failed"
Corrstat2. ="Failed" ?
Y
N
Do Nothing
Corrstat3. ="Failed" ?
Y
N
Do Nothing
Override Event severity and Node status
2nd field in 1st Attr. ="recovered" ?
Y
N
Set Attr. CorrStat1 "Resolved"
Do Nothing
4516/451606

Figure 94. Flowchart of Set State Example

To begin we will first define a new trap in NetView for AIX. This can be accomplished by completing the following steps.

1. First access the NetView for AIX Event Configuration function by selecting **Options**→**Event Configuration**→**Trap Customization** from the menu bar. Figure 95 on page 126 shows the Event Configuration window.



Enterprise Name	Enterprise ID
LNM_for_AIX	1.3.6.1.4.1.2.6.21.7.1
hmp6000	1.3.6.1.4.1.2.6.40
ibm7137	1.3.6.1.4.1.2.6.51
ibm_win_nv	1.3.6.1.4.1.2.6.87
ITSO-Raleigh	1.3.6.1.4.1.2.8.1
ibm	1.3.6.1.4.1.2
cisco	1.3.6.1.4.1.9
hp9000_300	1.3.6.1.4.1.11.2.3.2.2

Event Name	Event	Severity	Status	Sources
appl_alert	Specific 1	Warning	Default	

Figure 95. Event Configuration Window

2. On this window you will see that we have already added the ITSO-Raleigh experimental enterprise ID. You could use the NetView/6000 enterprise if you want to generate your own traps. Now select the enterprise and click on **Add**. This will result in the Add Event window shown in Figure 96 on page 127.

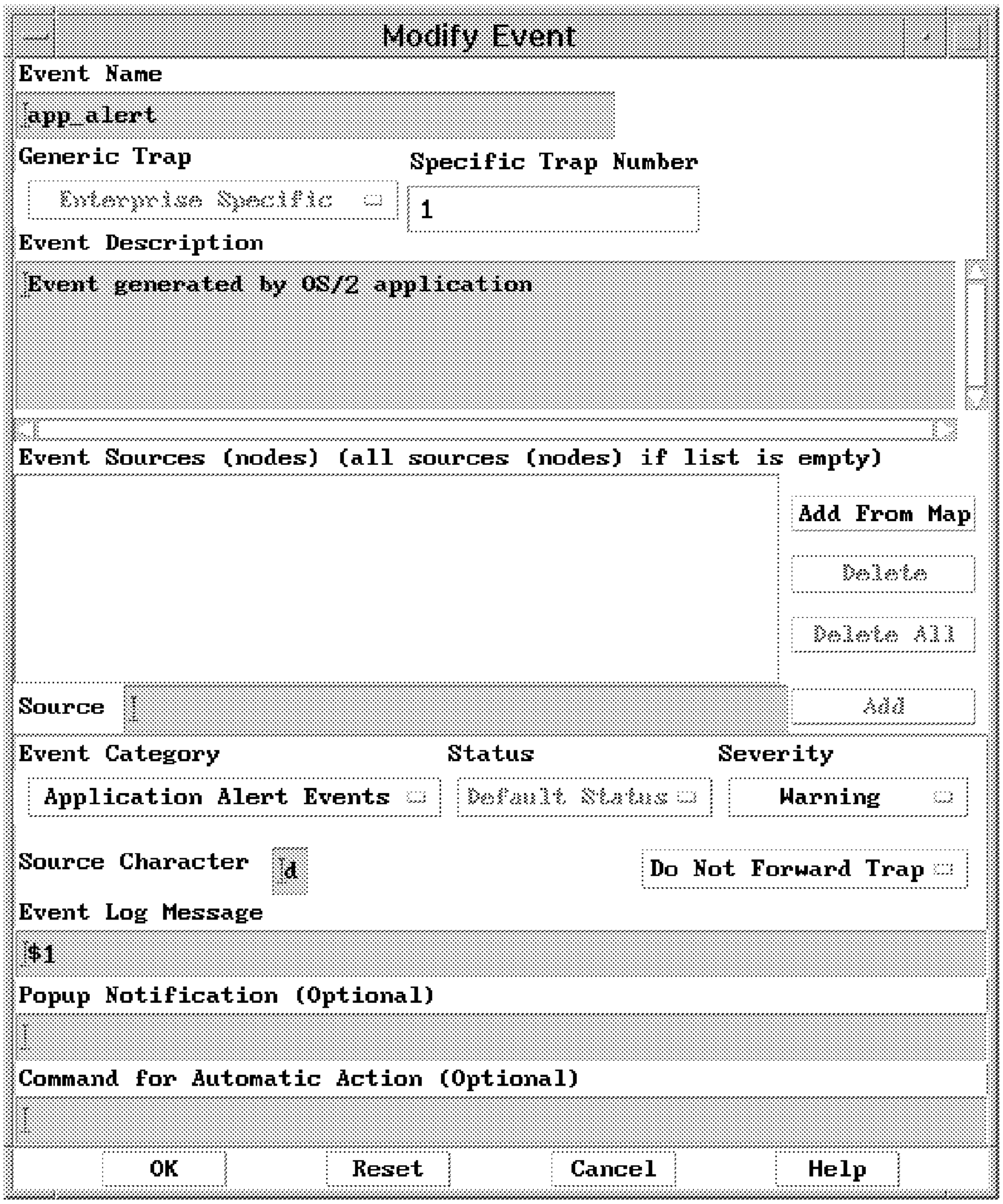


Figure 96. Editing Event Attributes

3. In this window we set the following attributes:

Event Name	app_alert
Specific Trap Number	1
Event Category	Application Alert Events
Source Character	d
Event Log Message	$1 (this means: put the first variable from the trap in the event display)

Now we are ready to build the ruleset for this example. The completed ruleset is shown in Figure 97 on page 128.

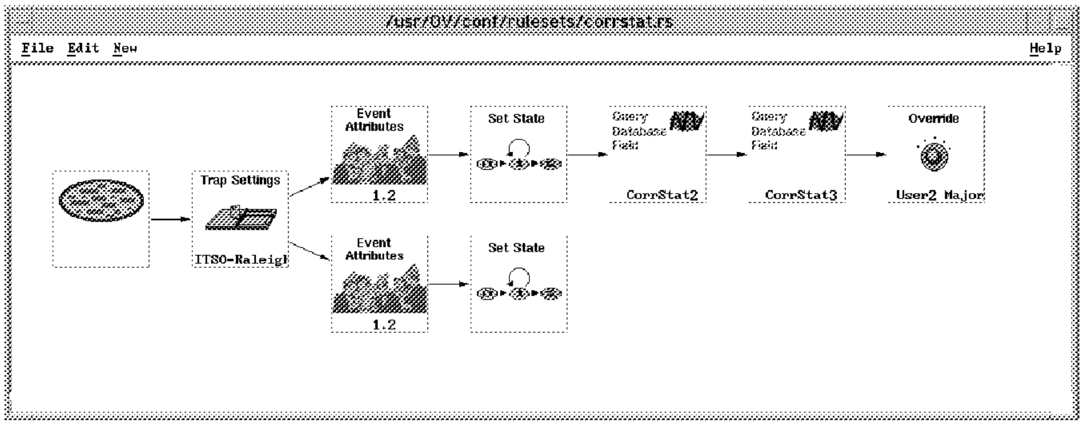
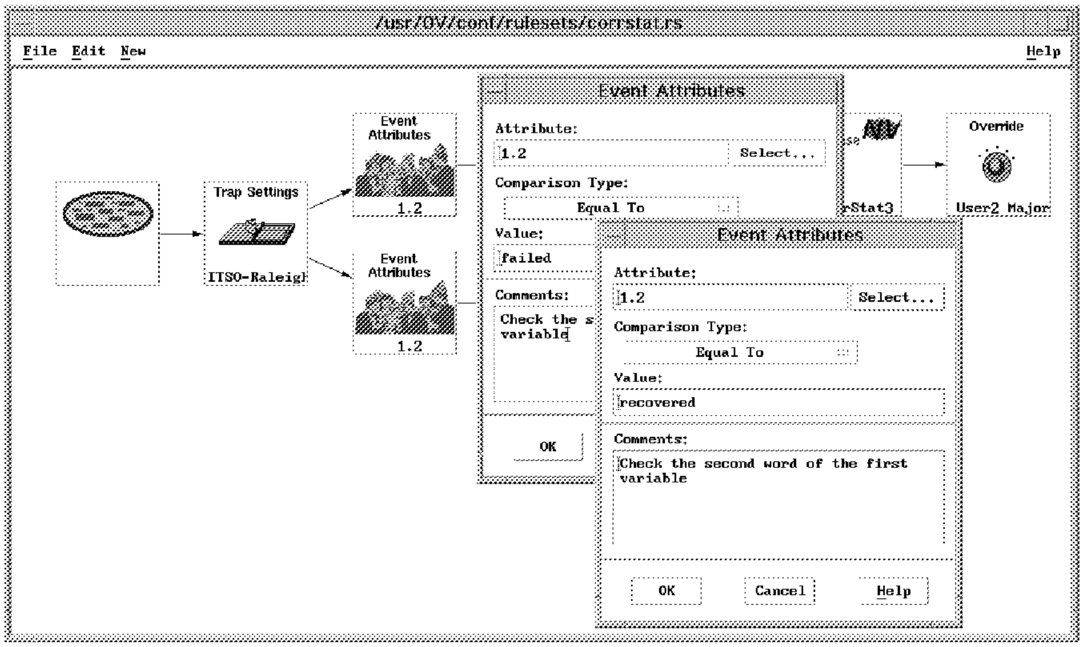


Figure 97. Complete Ruleset for the Set State Example

This ruleset was constructed by completing the following steps:

1. First we set the default behavior for the ruleset by double-clicking the event stream node (see Figure 59 on page 89).
2. Next we identify our trap with a Trap Settings node.
3. We then want to detect whether the trap being sent is a failure message or a recovery message. The application in this case always sends the same trap, with the meaning contained in the text of the first variable. This means that we use Event Attributes nodes to test, not the total message, but a word from within that message. The first Event Attributes node looks for the second word of the variable to be ″failed″ and the second looks for ″recovered″. This is illustrated in Figure 98.



Attribute: 1.2
Comparison Type: Equal To
Value: failed
Comments: Check the s variable

Attribute: 1.2
Comparison Type: Equal To
Value: recovered
Comments: Check the second word of the first variable

Figure 98. Adding the Event Attributes Nodes

In this dialog, we set the following fields:

Attribute	1.2 (meaning that we will compare the second word of the first variable inside the trap)
Comparison Type	Equal To
Value	failed or recovered

4. We then add Set State nodes to set the value of corrstat1 to Failed or Resolved, depending on which message type we detected. Figure 99 shows how to configure these nodes. Note that the Object ID Source field is set to Origin. This means that the ruleset processor will try to update the corrstat1 field in the object database record for the node from which the trap came.

Setting the Origin ID

Several of the ruleset nodes ask you to specify an Object ID Source. This specifies the object database record that it will refer to. For events that come from NetView for AIX itself, or from the Systems Monitor Mid-Level Manager the object node is a variable in the trap, usually variable 2. For traps that actually originate directly from a managed node, as in this case, the trap Origin should be chosen.

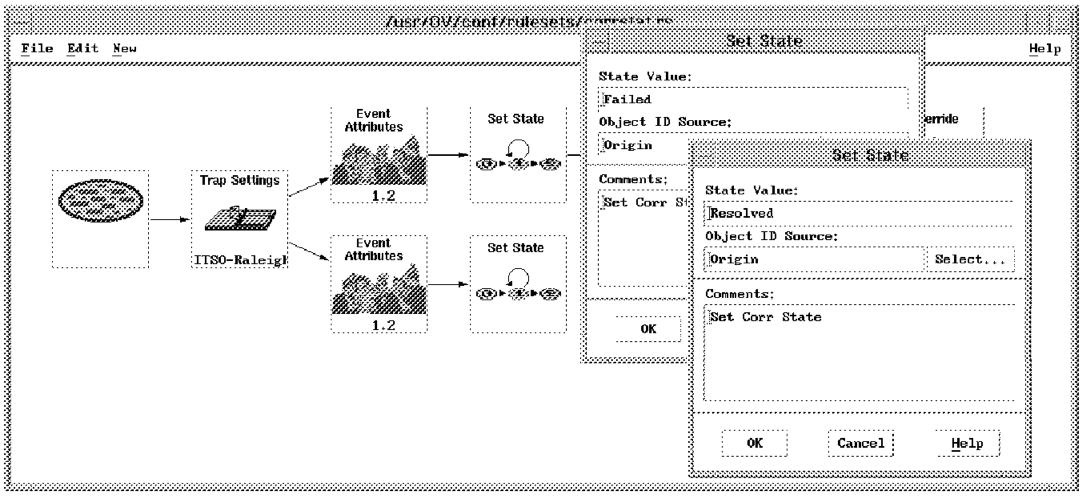
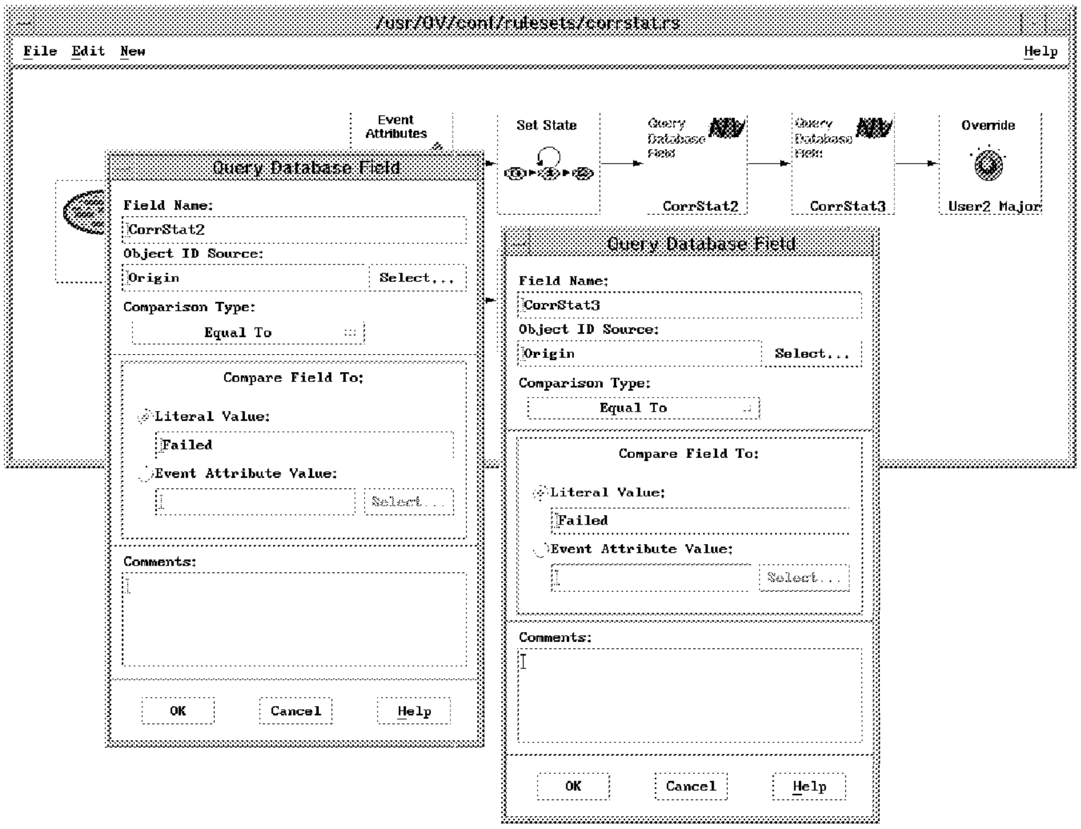


Figure 99. Adding the Set State Nodes

5. Finally we want to check to see if the application failure message has occurred three times in a row and if so, take some action. Each time we update corrstat1 using the Set State node, the previous value is moved into corrstat2 (and the previous value of corrstat2 is passed into corrstat3, etc.). To perform our test we therefore need to check corrstat2 and corrstat3. The Query Database Field node allows us to do this. Figure 100 on page 130 shows the configuration for this and also the Override node which will set the status of the affected node to User2 and the severity of the event to Major.



Query Database Field

Field Name: CorrStat2
Object ID Source: Origin
Comparison Type: Equal To
Compare Field To:
Literal Value: Failed
Event Attribute Value:
Comments:

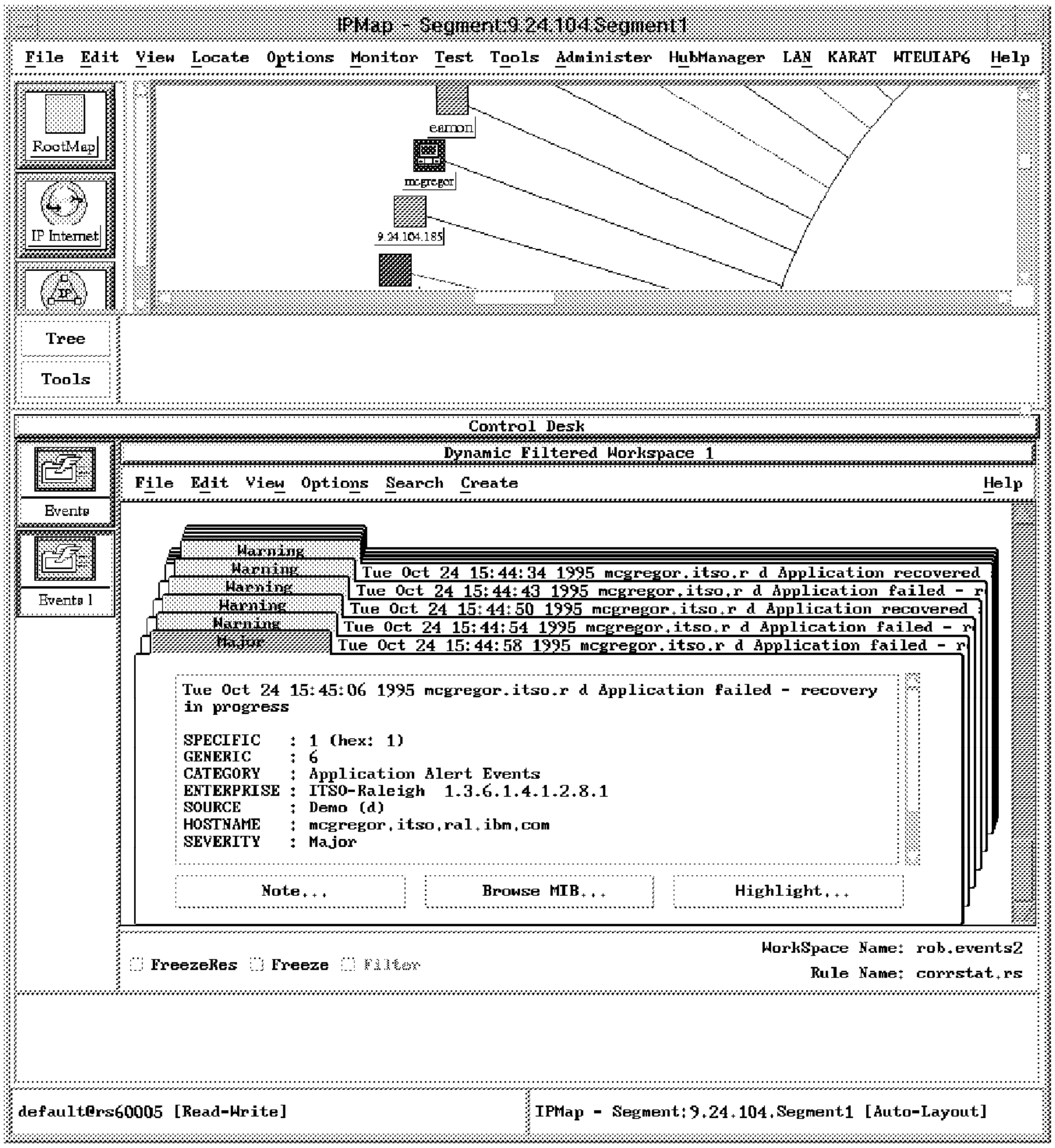
Query Database Field

Field Name: CorrStat3
Object ID Source: Origin
Comparison Type: Equal To
Compare Field To:
Literal Value: Failed
Event Attribute Value:
Comments:

Figure 100. Checking for Repeated Failures

6.2.6.1 Testing the Ruleset

1. First, we opened the workspace with the correct ruleset file.
2. Then, we sent the application alert traps. We used a locally-written OS/2 program to generate a simple trap (you can get a copy of this program via anonymous FTP, see Appendix A, “How to Obtain the Samples in this Book” on page 257 for details). We could equally well have used the snmptrap command provided by NetView for AIX or Systems Monitor for AIX to generate the trap from AIX.
3. The result of sending a failure event, followed by a recovery event followed by three failure events is shown in Figure 101 on page 131.

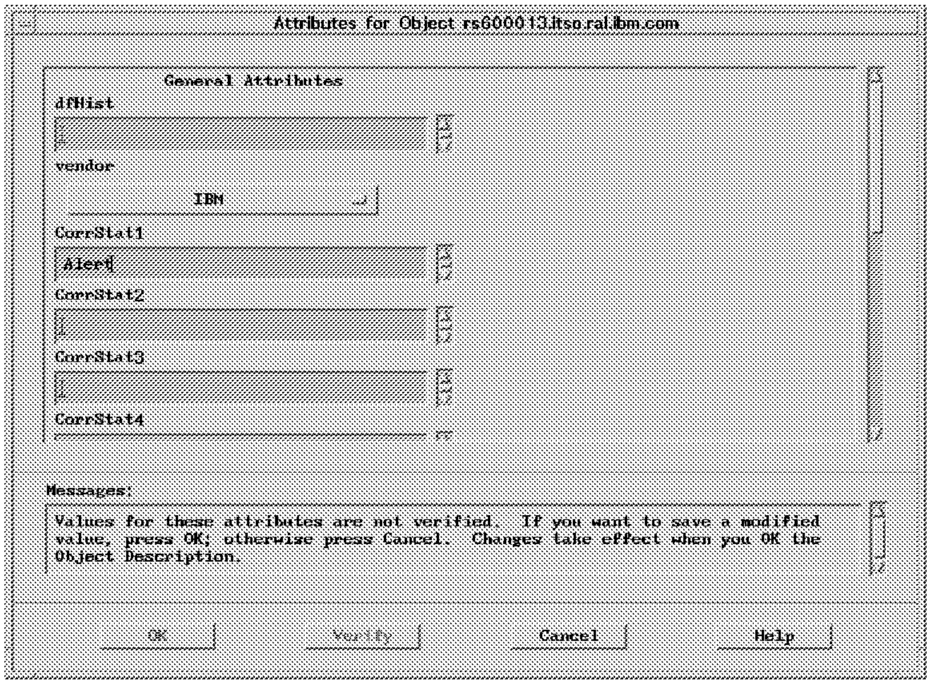

IPMap - Segment:9.24.104.Segment1
File Edit View Locate Options Monitor Test Tools Administer HubManager LAN KARAT WTEUIAP6 Help
RootMap
IP Internet
eamon
mcgregor
9.24.104.185
Tree
Tools
Control Desk
Events
Events 1
Dynamic Filtered Workspace 1
File Edit View Options Search Create Help
Warning
Warning
Warning
Warning
Warning
Major

```
Tue Oct 24 15:44:34 1995 mcgregor.itso.r d Application recovered
Tue Oct 24 15:44:43 1995 mcgregor.itso.r d Application failed - r
Tue Oct 24 15:44:50 1995 mcgregor.itso.r d Application recovered
Tue Oct 24 15:44:54 1995 mcgregor.itso.r d Application failed - r
Tue Oct 24 15:44:58 1995 mcgregor.itso.r d Application failed - r

Tue Oct 24 15:45:06 1995 mcgregor.itso.r d Application failed - recovery
in progress

SPECIFIC   : 1 (hex: 1)
GENERIC    : 6
CATEGORY   : Application Alert Events
ENTERPRISE : ITSO-Raleigh  1.3.6.1.4.1.2.8.1
SOURCE     : Demo (d)
HOSTNAME   : mcgregor.itso.ral.ibm.com
SEVERITY   : Major
```

```
WorkSpace Name: rob.events2
Rule Name: corrstat.rs
default@rs60005 [Read-Write]
IPMap - Segment:9.24.104.Segment1 [Auto-Layout]
```

Figure 101. The Workspace after Application Event Stream

4. We can also check to see if the value for corrstat1 was set. This can be accomplished by selecting the node on the map and clicking on **Edit->Modify/Describe->Object...** from the menu bar and then selecting General Attributes and clicking on **View/Modify**. The resulting Object Attributes window is shown in Figure 102 on page 132.


Attributes for Object rs600013.itso.ral.ibm.com

General Attributes

dfHist

vendor: IBM

CorrStat1: Alert

CorrStat2

CorrStat3

CorrStat4

Messages:

Values for these attributes are not verified. If you want to save a modified value, press OK; otherwise press Cancel. Changes take effect when you OK the Object Description.

Figure 102. The Object Attribute Window

6.2.7 Setting Database Fields

As we saw in the last example, event rulesets allow you to set the special purpose database fields, corrstat1 through corrstat5. They also provide the ability to set other database fields. This is a powerful function because not only can you set fields that are already defined in the database, but you can also set fields that you have defined yourself with field registration files or using NetView for AIX APIs. We used this function to set a field that we created ourselves. The field we created was defined according to information contained in a NetFinity trap. If we received a trap from NetFinity that reported that a NetFinity manager was offline, our field, called nfstatus, was set to express this state. This field was then changed to Online if the manager was reported to be back online. Figure 103 displays a flowchart of what we defined with this example.

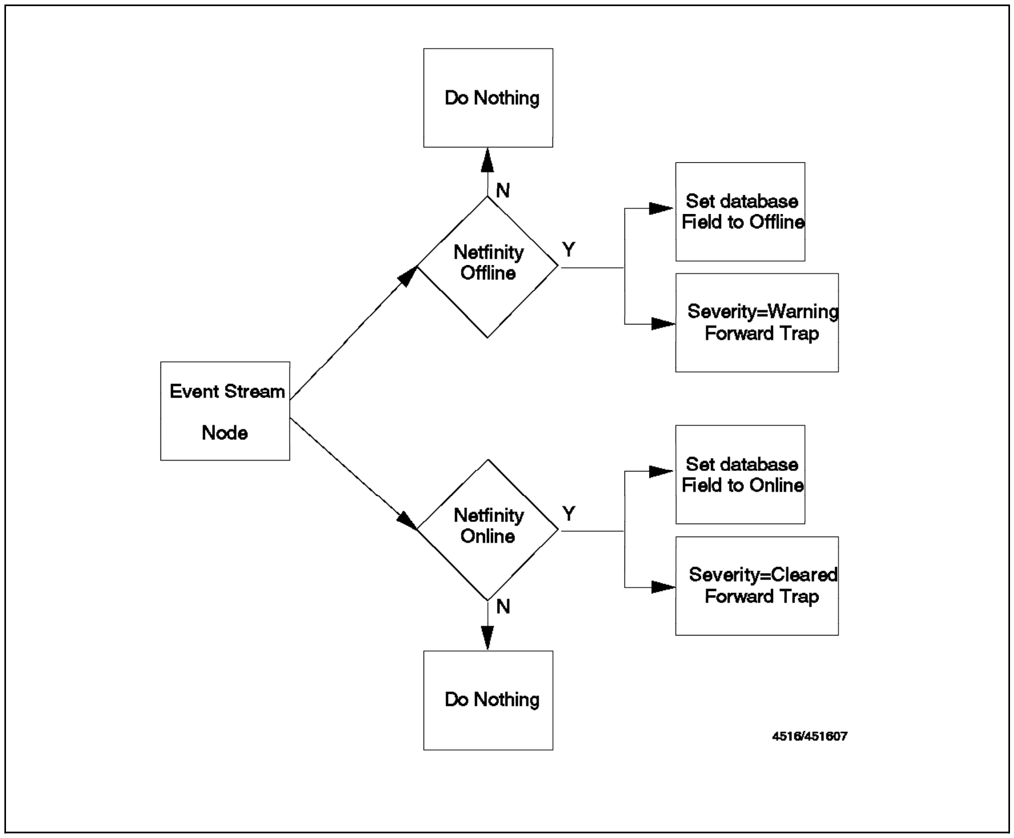

Do Nothing
N
Netfinity Offline
Y
Set database Field to Offline
Severity=Warning Forward Trap
Event Stream Node
Set database Field to Online
Netfinity Online
Y
Severity=Cleared Forward Trap
N
Do Nothing
4516/451607

Figure 103. Flowchart of Setting Database Fields Example

Our first step for this example was to create a database field. The easiest way to do this is to create a field registration file in directory /usr/OV/fields/C and then run command `nv6000 -fields`. This procedure is described in *NetView for AIX Programmer's Guide*, SC31-8164 (also available as one of the NetView for AIX online books).

In our case, however, we elected to use the NetView for AIX ovwdb API to create the field. This turned out to be a relatively straightforward procedure. The following steps were needed to add the field:

1. We wrote and compiled a small C program that allows you to add a field by issuing a simple command. The program is shown in Figure 104 on page 134.

```
/*---------------------------------------------------------------------
   crea_set_field.c
   Program to create and set a field value to something
   Arguments: name of the object, name of the field, value of the field
---------------------------------------------------------------------*/
#include <stdio.h>
#include <OV/ovw.h>
#include <OV/ovw_obj.h>

void main( int argc, char **argv)
{
char            *fieldName ;
OVwFieldId      newfieldId ;
char            *selectionName ;
char            *value ;
OVwObjectId     objectId ;

selectionName = argv[1] ;
fieldName = argv[2] ;
value = argv[3] ;
OVwDbInit() ;
newfieldId = OVwDbCreateField( fieldName, ovwStringField, ovwGeneralField ) ;
printf( "Field ID is %d\n", newfieldId ) ;
objectId = OVwDbSelectionNameToObjectId( selectionName ) ;
printf( "Object ID is %d\n", objectId ) ;
OVwDbSetFieldStringValue( objectId, newfieldId, value ) ;
}
```

Figure 104. C Program to Add A Field to the Object Database

2. After compiling this program, we were able to add a field called nfstatus into the database by entering the following command into an AIX command window:

   ```
   crea_set_field netfos2.itso.ral.ibm.com nfstatus unknown
   ```

3. To check that the field was created, we entered the following command:

   ```
   /usr/OV/bin/ovobjprint -s netfos2.itso.ral.ibm.com
   ```

   The results of entering this command are shown in Figure 105 on page 135.

```
                         OBJECTID                      SELECTION NAME


OBJECT: 2058

        FIELD ID         FIELD NAME                     FIELD VALUE
        10               Selection Name                 "netfos2.itso.ral.ibm.com"
        11               IP Hostname                    "netfos2.itso.ral.ibm.com"
        14               OVW Maps Exists                2
        15               OVW Maps Managed               2
        19               IP Status                      Normal(2)
        22               isIPRouter                     FALSE
        34               vendor                         Unset(0)
        44               isNode                         TRUE
        46               isComputer                     TRUE
        47               isConnector                    FALSE
        48               isBridge                       FALSE
        49               isRouter                       FALSE
        50               isHub                          FALSE
        68               isIP                           TRUE
        84               isSNMPSupported                TRUE
        86               SNMP sysDescr                  "OS/2 SNMP AGENT version 1.2
        87               SNMP sysLocation               "Building 657"
        88               SNMP sysContact                "Hermann"
        89               SNMP sysObjectID               "1.3.6.1.4.1.2.6.46"
        90               SNMPAgent                      Unset(0)
        94               isMLM                          FALSE
        95               isSYSMON                       FALSE
        96               isSIA                          FALSE
        97               isManager                      FALSE
        99               isSLM                          FALSE
        100              isSIAOS2                       FALSE
        104              TopM Interface Count           1
        110              TopM Interface List            "802.5  Up 9.24.104.114
        171              XXMAP Protocol List            "IP"
        200              IP Name                        "netfos2.itso.ral.ibm.com"
        352              default IP Symbol List         367
        1035             randy IP Symbol List           294
        3533             RouterStatus                   "Up"
        3545             nfstatus                       "unknown"
```

Figure 105. Results of ovobjprint Command. Our new nfstatus field is at the bottom of the list.

Our next task was to create a ruleset that was able to set this field. Figure 106 on page 136 displays the completed ruleset that we constructed.

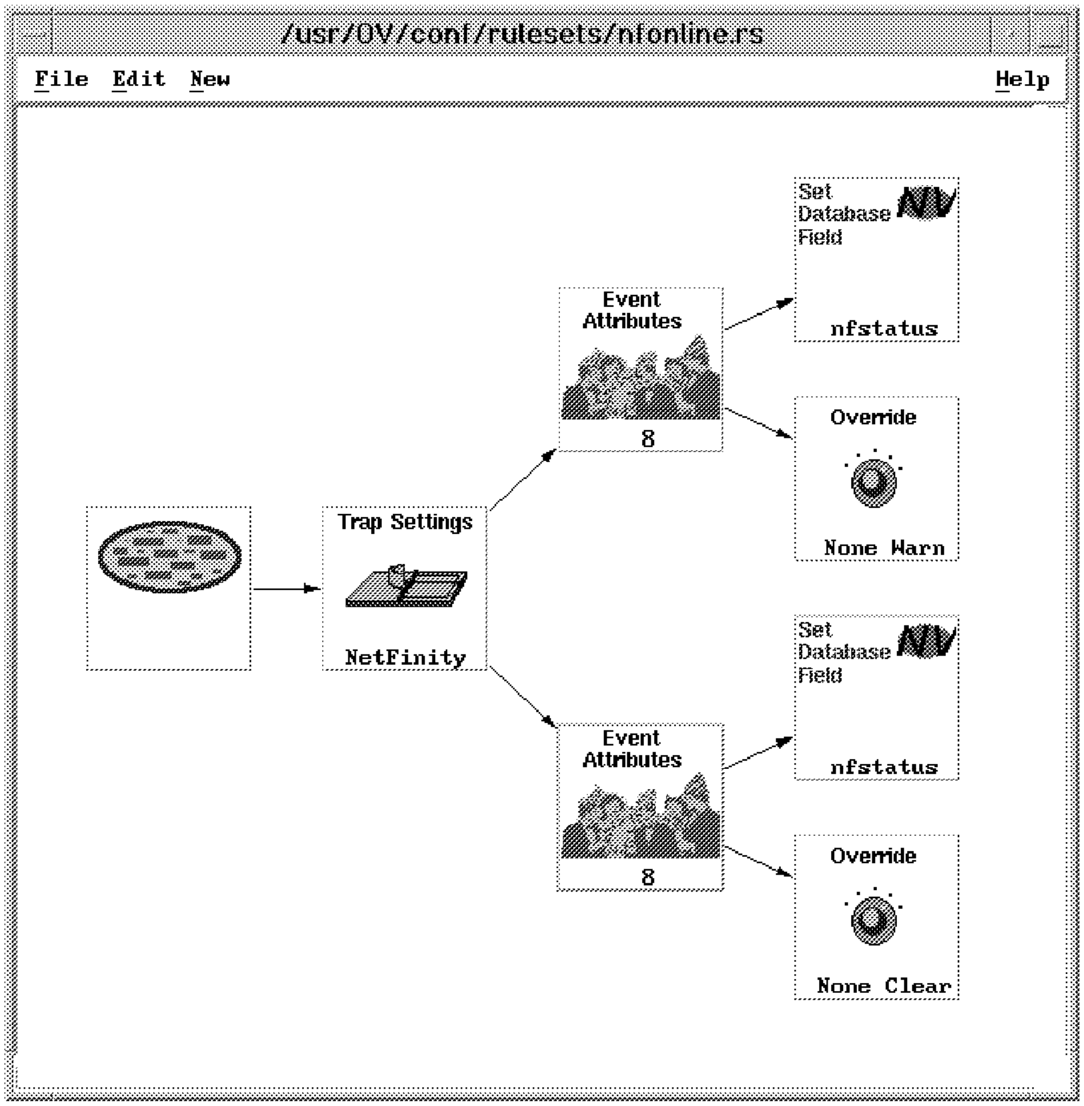
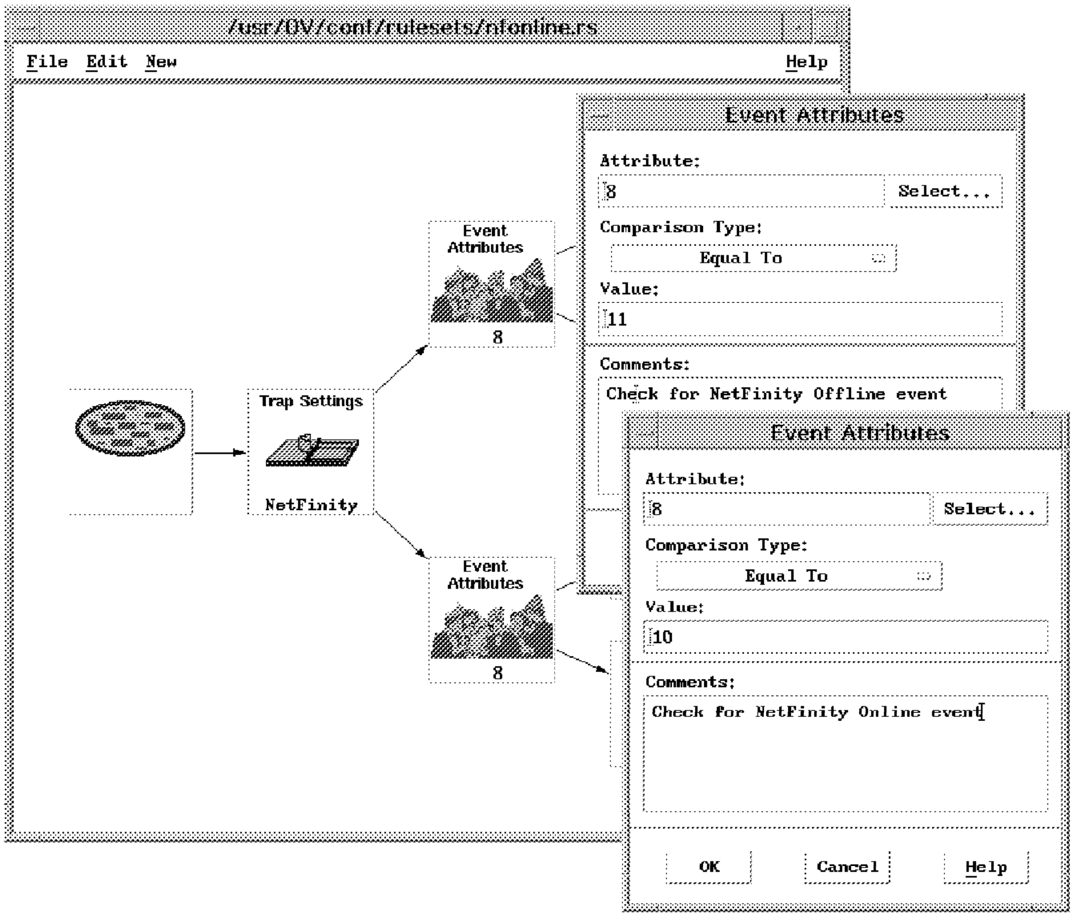


Figure 106. Rule that Sets an Object Database Field

To create this ruleset, we completed the following steps:

1. As always the first step was to set the default behavior (to Pass in this case) by double-clicking the event stream node.

2. In this example, we wanted to identify when NetFinity indicated that a system had gone offline and when it had gone online. To accomplish this, we added a Trap Settings node and a Event Attributes node for each type of trap. NetFinity uses one trap type for all events, so only one Trap Settings node is needed. The Event Attributes nodes were checking the value of the eighth parameter in the trap. For NetFinity traps, the eighth parameter is equal to the trap type. If trap type was equal to 11, we knew that a NetFinity system had gone offline. If trap type was equal to 10, then we knew that the system was online. Figure 107 on page 137 illustrates how we identified these traps.

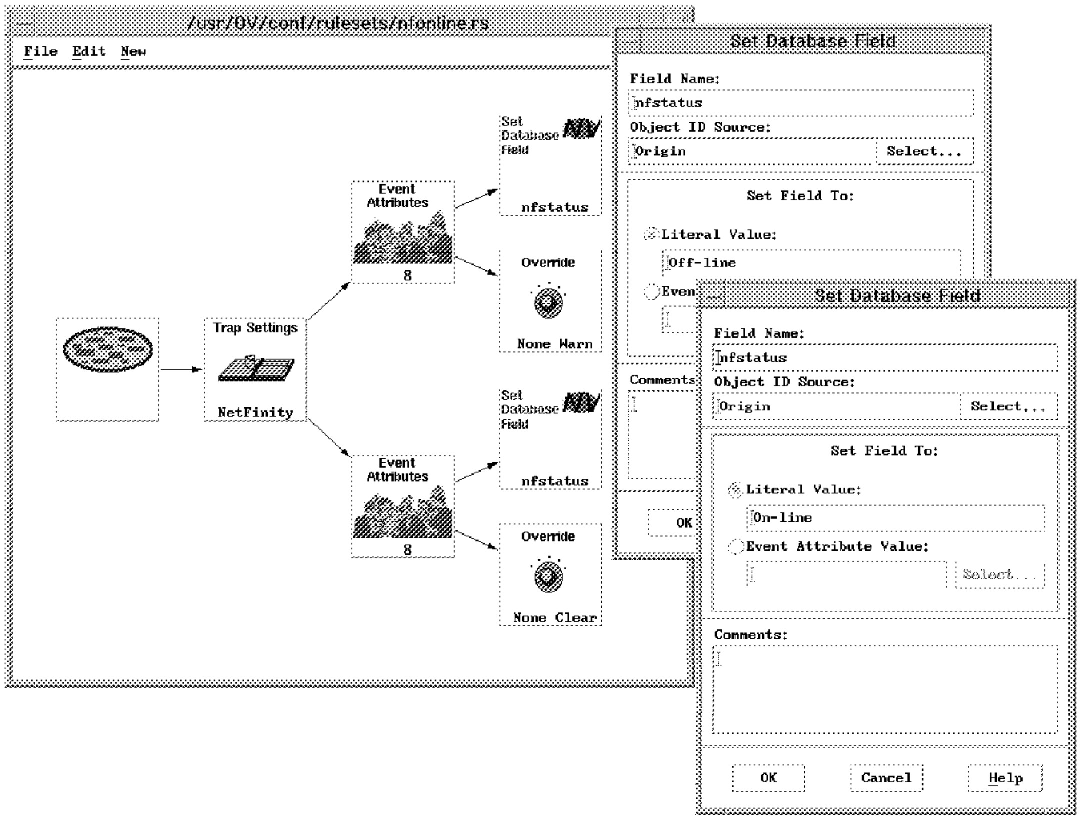


Attribute: 8
Comparison Type: Equal To
Value: 11
Comments: Check for NetFinity Offline event

Attribute: 8
Comparison Type: Equal To
Value: 10
Comments: Check for NetFinity Online event

Figure 107. Identifying NetFinity Trap Types

3. To each trap pathway, we then added a Set Database Field node. We configured each of these nodes to set our database field called nfstatus to either Offline or Online, depending on the pathway, as shown in Figure 108 on page 138.



Set Database Field

Field Name: nfstatus

Object ID Source: Origin Select...

Set Field To:

Literal Value: Off-line

Event

Comments:

OK

Set Database Field

Field Name: nfstatus

Object ID Source: Origin Select...

Set Field To:

Literal Value: On-line

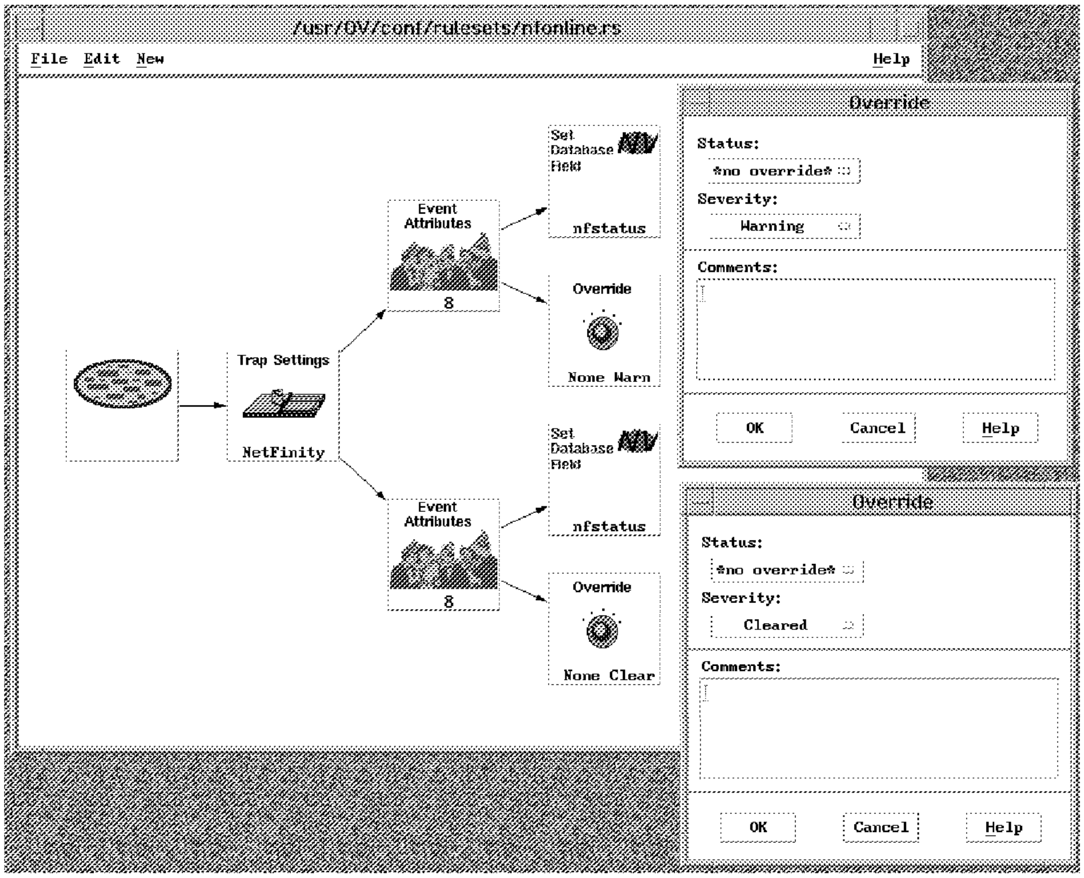
Event Attribute Value: Select...

Comments:

OK Cancel Help

Figure 108. Configuring the Set Database Field Dialog

4. We also attached an Override node to each trap pathway. These were configured to make traps of type 11 change the severity of the trap to Warning, and to make traps of type 10 change the severity to Cleared. Figure 109 on page 139 shows how this was accomplished.



Override

Status: *no override*

Severity: Warning

Comments:

Override

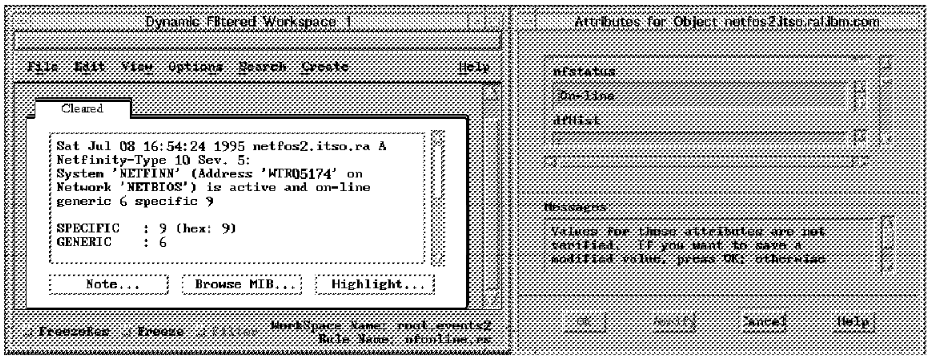
Status: *no override*

Severity: Cleared

Comments:

Figure 109. Overriding the Severity of NetFinity Traps

We were able to test our ruleset by receiving traps from NetFinity that reported on the online status of other NetFinity systems. One quick way of determining if the database field had changed was to select the object and view database settings by selecting Edit→Modify/Describe→Object... from the menu bar. This selection led us to the Object Selection box from which we chose to view the General Attributes. The window that resulted is shown in Figure 110. This figure also shows the trap that caused the database field to be set.


Dynamic Filtered Workspace 1
File Edit View Options Search Create Help
Cleared
Note... Browse MIB... Highlight...
FreezeRec Freeze Filter
Attributes for Object netfos2.itso.ral.ibm.com
Messages
OK Verify Cancel Help

```
Sat Jul 08 16:54:24 1995 netfos2.itso.ra A
Netfinity-Type 10 Sev. 5:
System 'NETFINN' (Address 'WTR05174' on
Network 'NETBIOS') is active and on-line
generic 6 specific 9

SPECIFIC   : 9 (hex: 9)
GENERIC    : 6
```

WorkSpace Name: root.events2
Rule Name: nfonline.rs

nfstatus: On-line

dfHist:

Values for these attributes are not verified. If you want to save a modified value, press OK; otherwise

Figure 110. NetFinity Event and Corresponding Database Field Setting

Having a database field that indicated whether a NetFinity system was online or not was useful to us. We were able to build collections that used this database field as part of a definition of nodes in trouble.

6.2.8 Setting Global Variables

In this example we revisit a previous example, 6.2.2, “Suppressing Events by Setting Thresholds” on page 101. In that case we were filtering out unwanted threshold exceptions by only forwarding them if they arrived at an unacceptably high frequency. In this example we assume that once per day we run a backup process on the server. While that process is running we *expect* high CPU utilization, so any thresholds can be ignored.

The solution we have chosen is to add commands to the start and end of the backup process which will send traps to NetView for AIX to indicate when the backups are running. We then modified our ruleset to set a global variable during the backup period and to suppress the event display based on its value.

Figure 111 displays a flow of what we want to obtain with this ruleset.

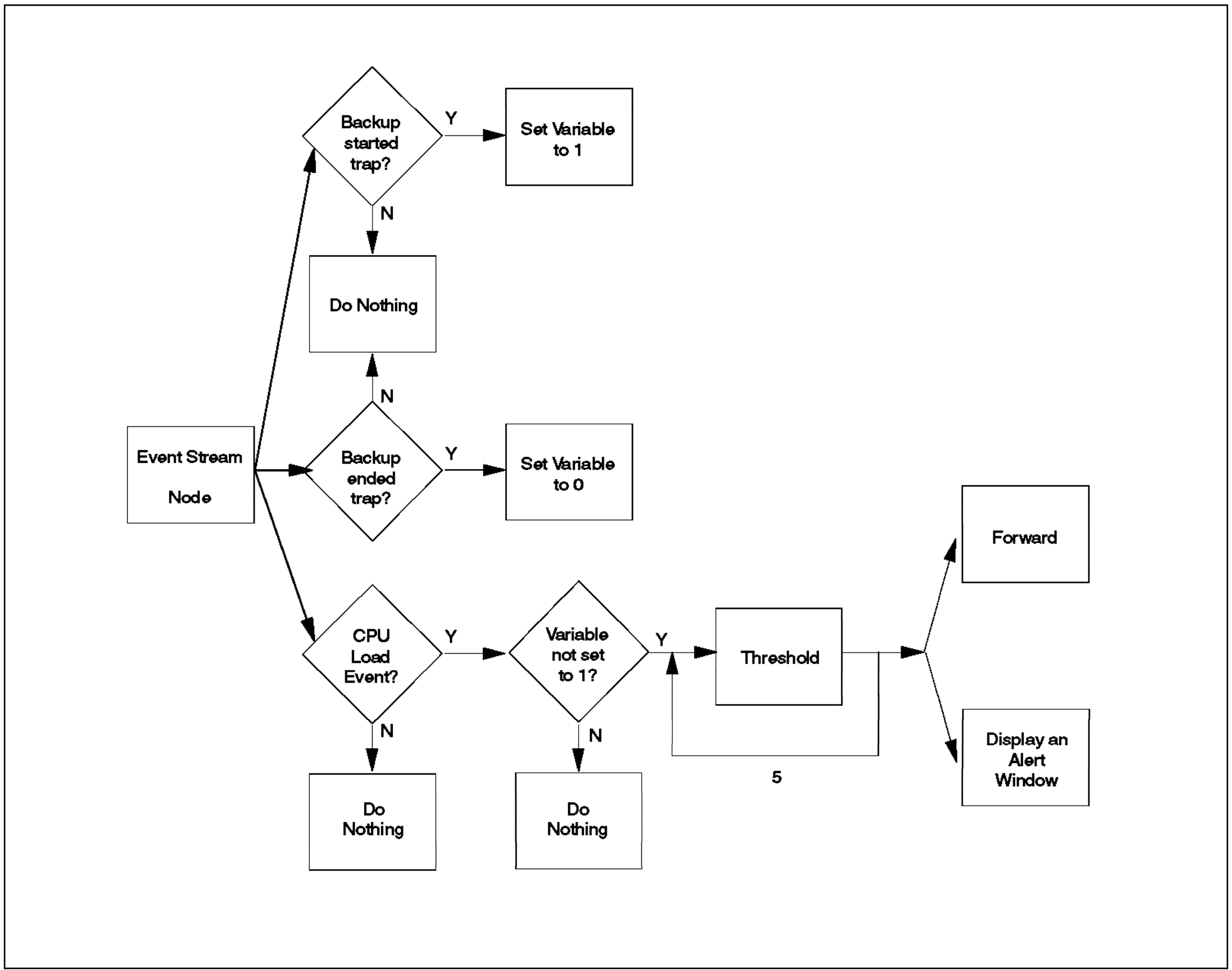


Figure 111. Flowchart of Setting Global Variables Example

The ruleset that we constructed to perform the task is shown in Figure 112 on page 141.

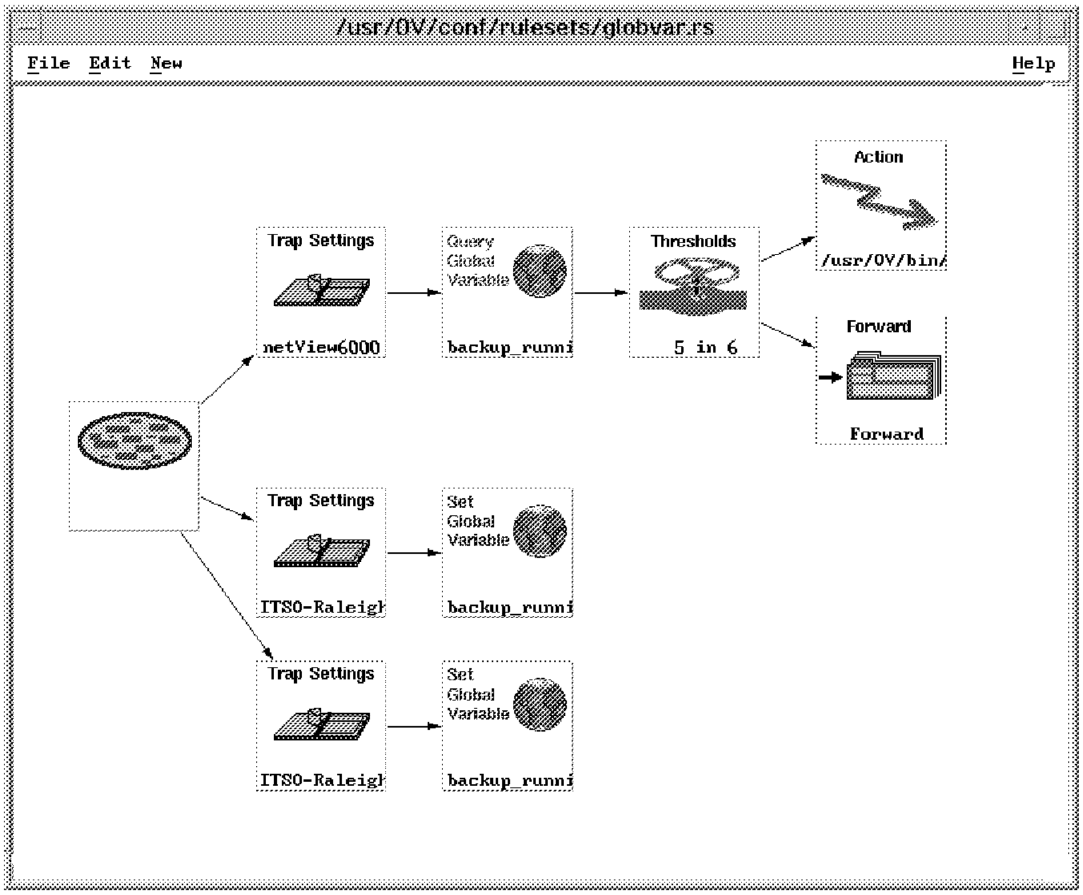
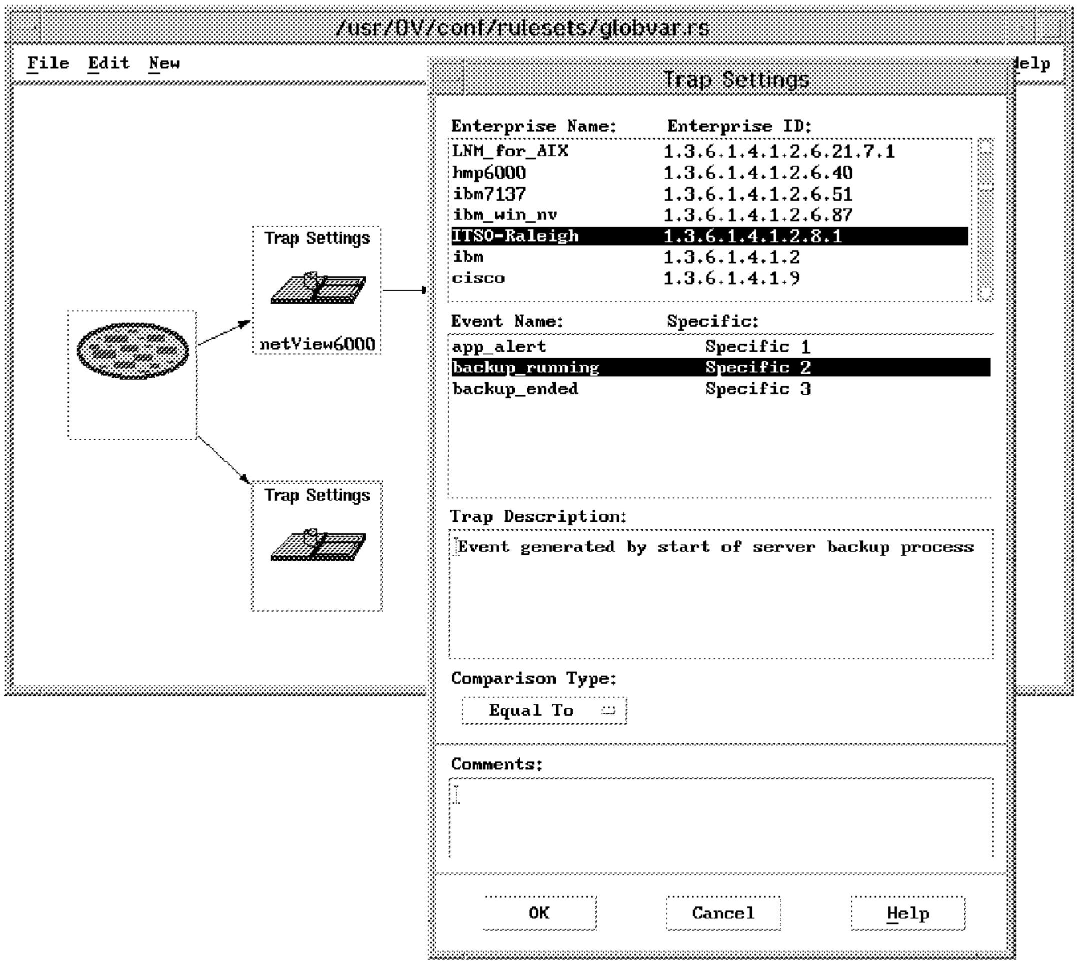


Figure 112. Completed Ruleset of Setting Global Variables Example

The first step in creating this ruleset was to define the new traps representing the start and end of the backup process. We again used our experimental enterprise ID and defined specific trap 2 for backup started and 3 for backup ended. Refer to Figure 95 on page 126 and Figure 96 on page 127 for details of how to add traps.

Having defined the traps, we could then go ahead and create the ruleset by doing the following:

1. As we were using the previous ruleset as a starting point, we started building the new ruleset by loading the old one into the editor. We did this by selecting File→Open from the menu bar and then choosing our threshold ruleset (thresh.rs) from the list. To avoid accidentally overwriting the previous ruleset we immediately saved it under a new name.
2. Next we added the Trap Settings node to check for the backup started event and connected it to the event stream node (see Figure 113 on page 142).



Trap Settings

Enterprise Name: Enterprise ID:

```
LNM_for_AIX      1.3.6.1.4.1.2.6.21.7.1
hmp6000          1.3.6.1.4.1.2.6.40
ibm7137          1.3.6.1.4.1.2.6.51
ibm_win_nv       1.3.6.1.4.1.2.6.87
ITSO-Raleigh     1.3.6.1.4.1.2.8.1
ibm              1.3.6.1.4.1.2
cisco            1.3.6.1.4.1.9
```

Event Name: Specific:

```
app_alert          Specific 1
backup_running     Specific 2
backup_ended       Specific 3
```

Trap Description:

Event generated by start of server backup process

Comparison Type: Equal To

Comments:

Figure 113. Adding the Trap Settings Node for Backup Started

3. We then added the Set Global Variable node, using the following parameters:

Variable Name	backup_running
Set Variable	we chose **Set to Literal Value** with the value **1**

In this case we are only using the global variable as a flag, so we used a single character (1) to represent the status. However, we could have placed detailed information in the variable, or taken information from the trap itself. Figure 114 on page 143 shows the dialog for adding this node.

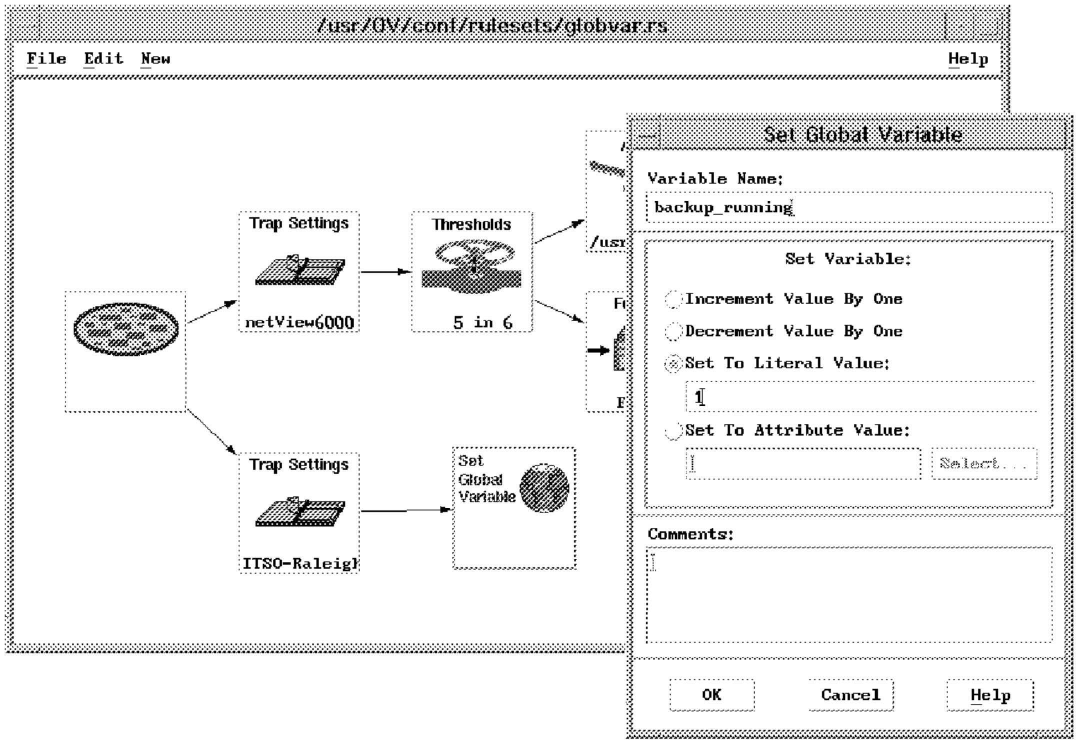
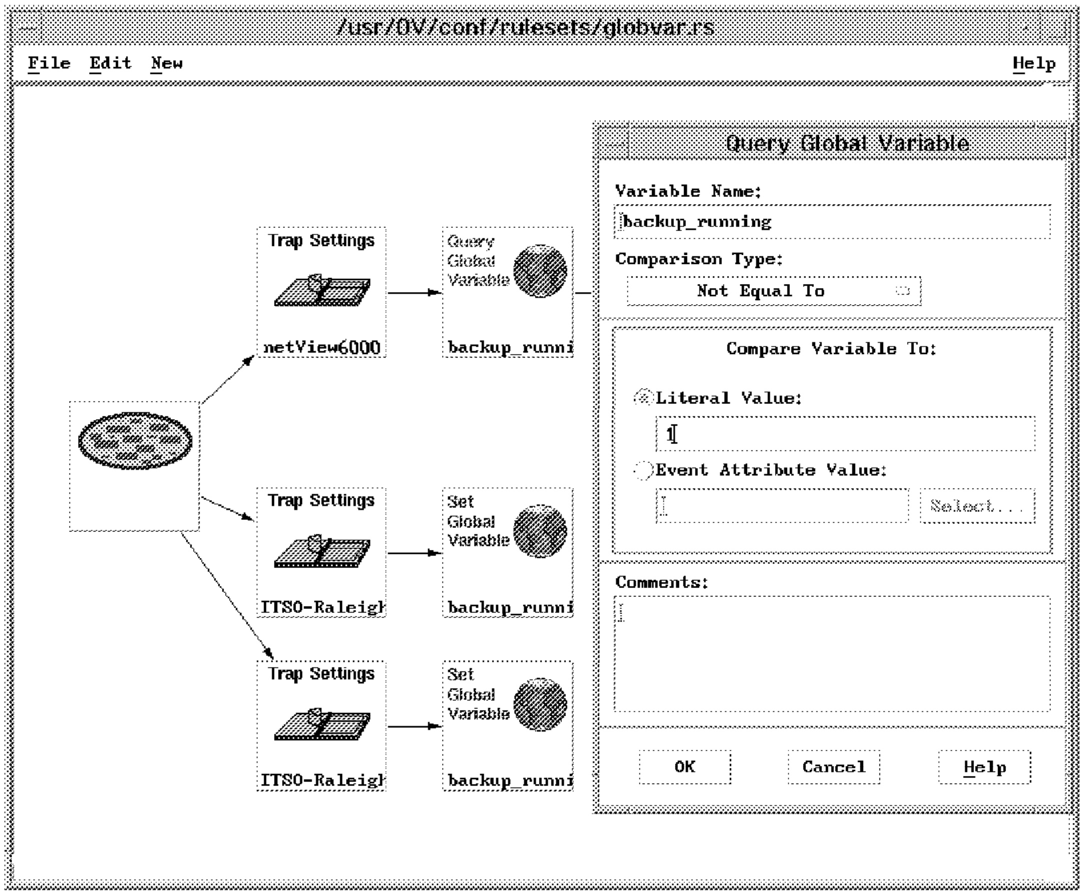


Figure 114. Adding the Set Global Variable Node

4. We then repeated the last two steps, but this time specifying the backup_ended trap, and setting the variable to zero.

5. Now we needed to modify the original part of the ruleset (inherited from the threshold example) to check for our global variable flag. To do this, we first removed the link between the Trap Settings node and the Thresholds node by selecting Edit→Delete Line from the menu bar. We then added a Query Global Variable node as shown in Figure 115 on page 144. This node will only be passed if the variable backup_running is *not* equal to 1 (that is, if there is *not* a backup running).



Variable Name: backup_running

Comparison Type: Not Equal To

Compare Variable To:

(●) Literal Value: 1

( ) Event Attribute Value:

Select...

Comments:

Figure 115. Adding the Query Global Variable Node

6. Finally we connected the Query Global Variable node into the ruleset as shown in Figure 112 on page 141.

You may wonder why we only have processing to pass the threshold events when a backup is not running, and nothing to suppress them when the backup *is* running. This is because we defined a default behavior of Block for the thresh.rs ruleset that this was based on. Note, however that this means that the backup_started and backup_ended traps will also not appear in the Event display, unless we add a Forward node to those legs of the ruleset.

Default Behavior: Should You Block or Pass?

In general, a default behavior of Pass is more ″natural″ for rulesets in an event display workspace. That is, anything that is not explicitly blocked by a ruleset Block Event Display or Thresholds node will be displayed to the user. One big advantage of this is that if some trap that has never been seen before should arrive, the user will know about it. The counter argument to this is that you are in danger of flooding the user with events that he does not need to see. We advocate that your policy should be to use a default behavior of Pass, but that you should do a thorough job of filtering and throttling unwanted events, to keep the event rate to an acceptable level. We discuss this further in Chapter 8, “Implementation Recommendations for Rulesets” on page 195.

For rulesets that are intended for automation only (that is, those added to /usr/OV/conf/ESE.automation) the question is moot, since the event is not passed on anywhere even if a Forward is indicated.

6.2.8.1 Testing the Ruleset

Now we can test if our ruleset works in the way we expect, using the following steps:

1. Open the workspace with the correct ruleset file, globvar.rs.
2. Send a trap from our server host to the NetView for AIX machine (rsserver) using the `snmptrap` command. The `snmptrap` command is provided by NetView for AIX which is not installed on the server. However, a version of the command is also provided by Systems Monitor for AIX SIA in the directory /usr/lpp/smsia/original. The trap we want to send is

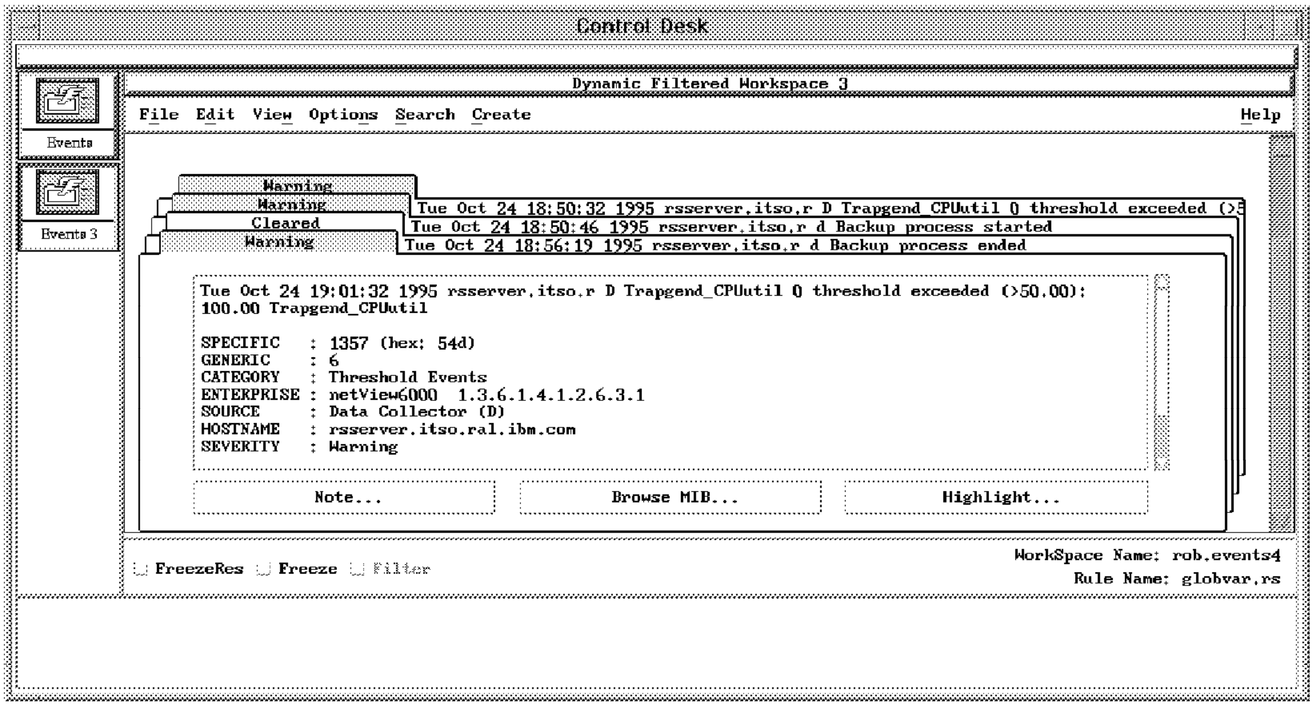
Enterprise ID	.1.3.6.1.4.1.2.8.1 (our ITSO experimental enterprise)
Generic Trap	6
Specific Trap	2 (backup_started)

   The command to do this is:

   ```
   /usr/lpp/smsia/original/snmptrap rs60005 public .1.3.6.1.4.1.2.8.1 `hostname` 6 2 0
   ```

   For the backup_ended trap use the same command with a specific trap ID of 3 instead of 2.

We tested the ruleset by sending these traps manually, with the expected results (CPU utilization events ceased after the backup started trap and recommenced after the backup ended trap). Figure 116 on page 146 shows the resulting event display.



```
Tue Oct 24 18:50:32 1995 rsserver.itso.r D Trapgend_CPUutil 0 threshold exceeded (>5
Tue Oct 24 18:50:46 1995 rsserver.itso.r d Backup process started
Tue Oct 24 18:56:19 1995 rsserver.itso.r d Backup process ended

Tue Oct 24 19:01:32 1995 rsserver.itso.r D Trapgend_CPUutil 0 threshold exceeded (>50.00):
100.00 Trapgend_CPUutil

SPECIFIC   : 1357 (hex: 54d)
GENERIC    : 6
CATEGORY   : Threshold Events
ENTERPRISE : netView6000  1.3.6.1.4.1.2.6.3.1
SOURCE     : Data Collector (D)
HOSTNAME   : rsserver.itso.ral.ibm.com
SEVERITY   : Warning

WorkSpace Name: rob.events4
Rule Name: globvar.rs
```

Figure 116. Workspace Showing Threshold and Backup Process Events. Note that we had to cheat a little to get this display, by adding Forward nodes to the backup_started and backup_ended paths in the ruleset. Otherwise they would have been suppressed by the default behavior of Block.

Normally you would add these commands to the backup shell script, or execute them as a job step using a scheduling package, such as SystemView Job Scheduler for AIX.

6.2.9 Setting MIB Variables

Another means by which rulesets allows you to change information about objects is through the Set MIB Variable function. We used this function to alter the polling interval of a Systems Monitor threshold when the threshold was in its breached state. Specifically, we shortened the polling time for a threshold that was set to monitor paging space. If paging space usage for a device exceeded 80%, a threshold arm trap was forwarded to NetView for AIX. When this trap arrived, a Set MIB Variable node was configured to reduce the polling time of the paging space threshold. When the threshold was resolved, the ruleset changed the polling time back to its original value. Figure 117 displays a flowchart of the trap processing.

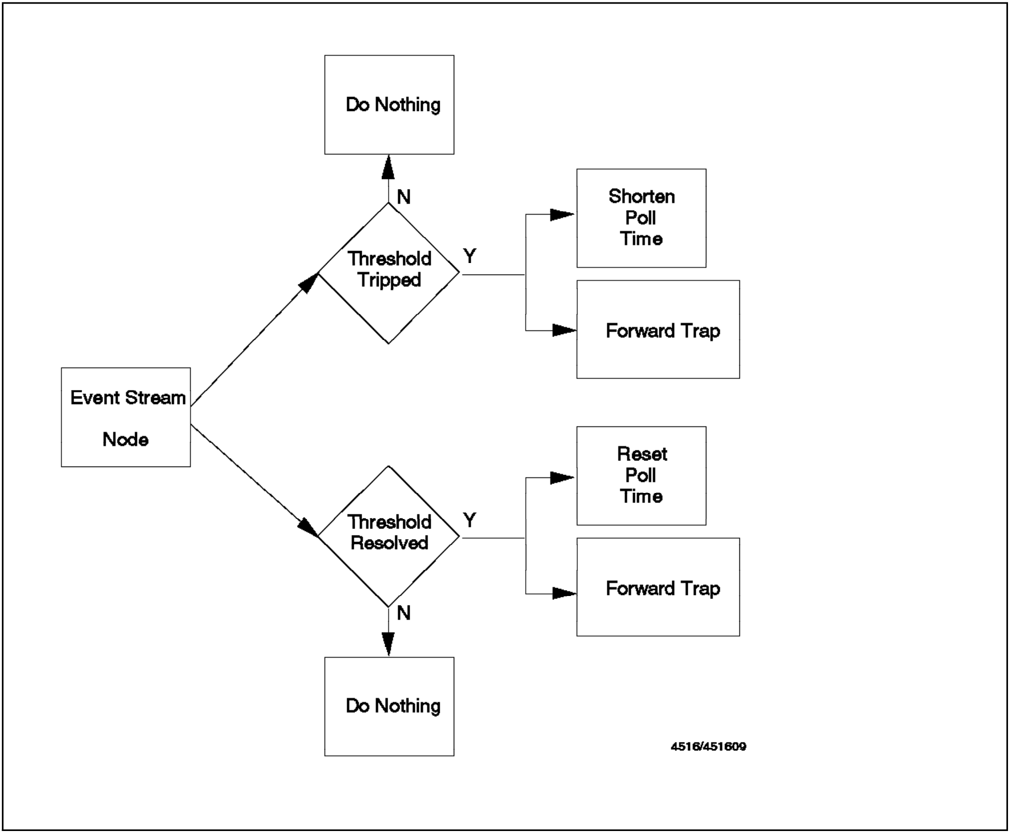


Figure 117. Flowchart of Set MIB Variable Example

Figure 118 on page 148 displays the ruleset that we constructed.

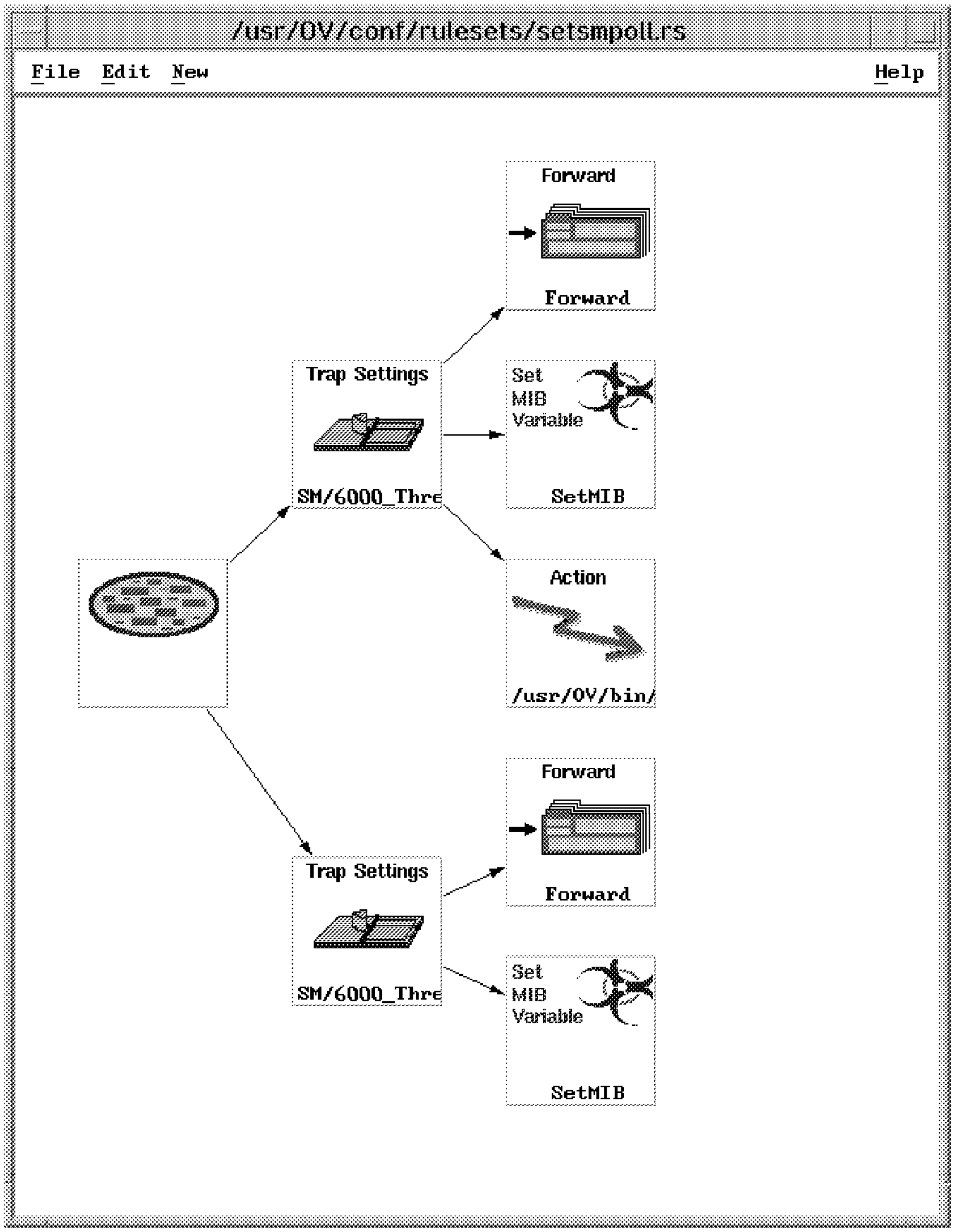


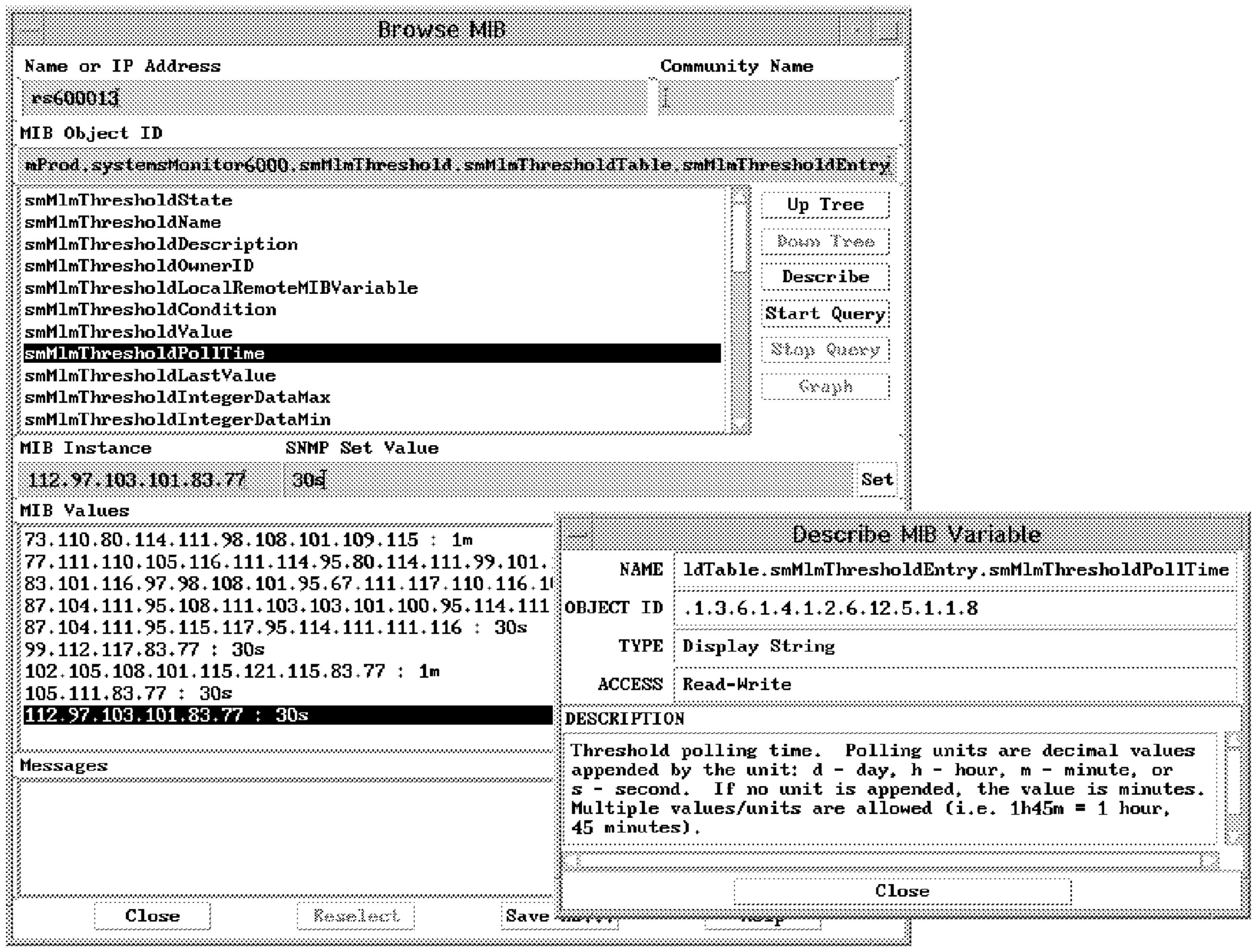
Figure 118. Rule that Shortens Polling Time when Threshold is in Breached State

The following steps describe how we constructed this ruleset:

1. We first identified the Systems Monitor Arm trap and Rearm trap by using the Trap Setting Nodes.

2. To each of these Trap Settings nodes we attached a Set MIB Variable node. In order to define the MIB information, we used the MIB Browser. To help us enter the MIB Variable Name, we stepped through the Systems Monitor MIB until we arrived at the smMlmThresholdPollTime variable. We then selected **Describe** to get the name and variable type. From the Describe function, we simply copied the name using the left mouse button and pasted it into the

MIB Variable Name field using the middle mouse button. To the end of the name we added the MIB instance which we were able to get by clicking on the **Start Query** function in the MIB Browser. Figure 119 on page 149 shows how the MIB Browser was used.



Name or IP Address: rs600013
Community Name:
MIB Object ID: mProd.systemsMonitor6000.smMlmThreshold.smMlmThresholdTable.smMlmThresholdEntry

```
smMlmThresholdState
smMlmThresholdName
smMlmThresholdDescription
smMlmThresholdOwnerID
smMlmThresholdLocalRemoteMIBVariable
smMlmThresholdCondition
smMlmThresholdValue
smMlmThresholdPollTime
smMlmThresholdLastValue
smMlmThresholdIntegerDataMax
smMlmThresholdIntegerDataMin
```

MIB Instance: 112.97.103.101.83.77
SNMP Set Value: 30s

MIB Values

```
73.110.80.114.111.98.108.101.109.115 : 1m
77.111.110.105.116.111.114.95.80.114.111.99.101.
83.101.116.97.98.108.101.95.67.111.117.110.116.1
87.104.111.95.108.111.103.103.101.100.95.114.111
87.104.111.95.115.117.95.114.111.111.116 : 30s
99.112.117.83.77 : 30s
102.105.108.101.115.121.115.83.77 : 1m
105.111.83.77 : 30s
112.97.103.101.83.77 : 30s
```

Messages

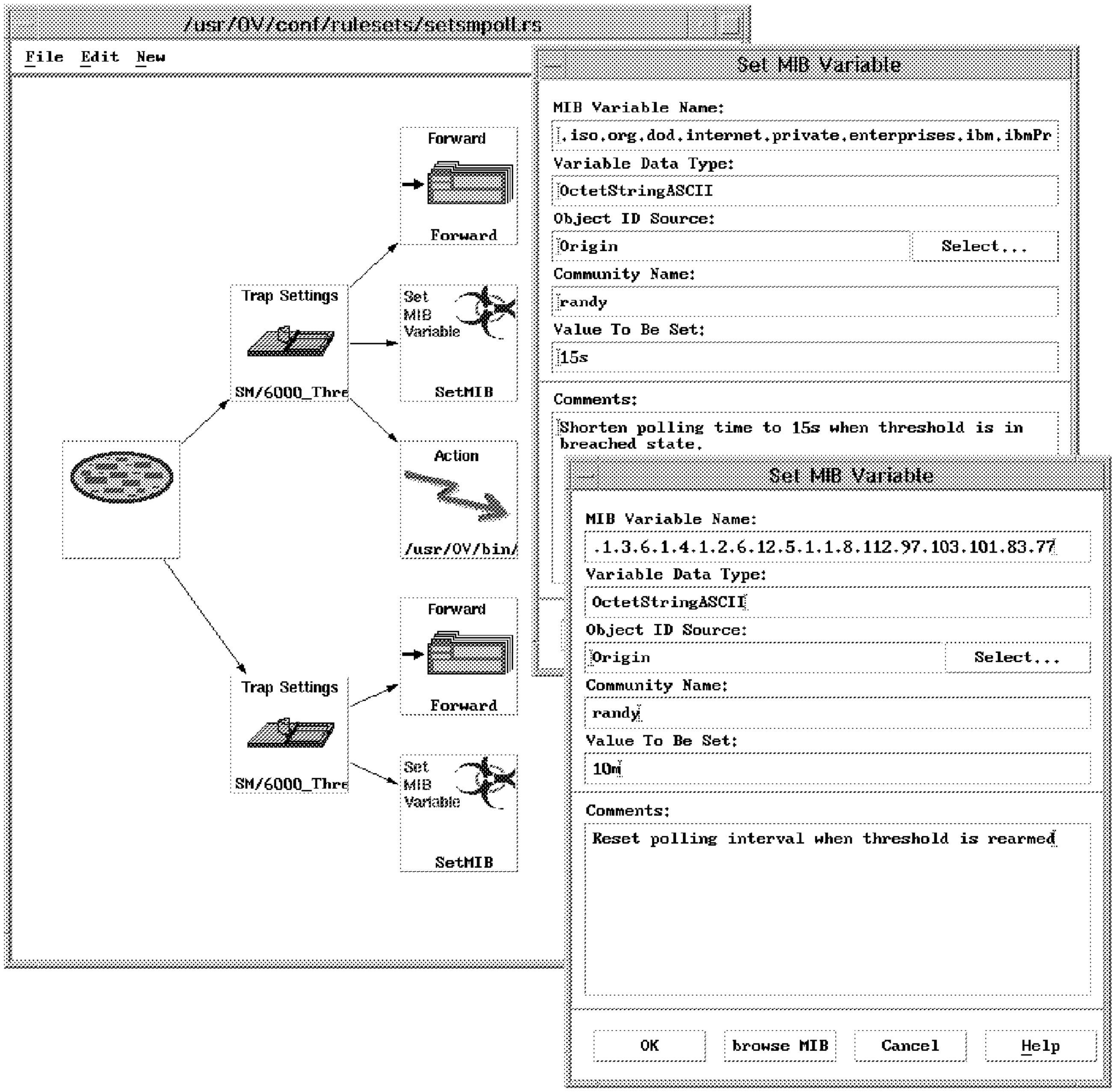
NAME	ldTable.smMlmThresholdEntry.smMlmThresholdPollTime
OBJECT ID	.1.3.6.1.4.1.2.6.12.5.1.1.8
TYPE	Display String
ACCESS	Read-Write

DESCRIPTION

Threshold polling time. Polling units are decimal values appended by the unit: d - day, h - hour, m - minute, or s - second. If no unit is appended, the value is minutes. Multiple values/units are allowed (i.e. 1h45m = 1 hour, 45 minutes).

Figure 119. Using the MIB Browser to Aid with Set MIB Variable Configuration

3. Note that when the trap reported that the threshold was breached (armed) we shortened the polling interval to 15 seconds. When the threshold was resolved (rearmed), we returned the polling time to 10 minutes. These settings are shown in Figure 120 on page 150.



MIB Variable Name: .iso.org.dod.internet.private.enterprises.ibm.ibmPr
Variable Data Type: OctetStringASCII
Object ID Source: Origin
Community Name: randy
Value To Be Set: 15s
Comments: Shorten polling time to 15s when threshold is in breached state.

MIB Variable Name: .1.3.6.1.4.1.2.6.12.5.1.1.8.112.97.103.101.83.77
Variable Data Type: OctetStringASCII
Object ID Source: Origin
Community Name: randy
Value To Be Set: 10m
Comments: Reset polling interval when threshold is rearmed

Figure 120. Configuring the MIB Variable Dialogs

6.2.10 Using Rulesets to Supplement Event Capabilities of Another Manager

One useful concomitant capability of ESE event rulesets is their ability to supplement the event management functions of other managers. Since NetView for AIX can receive traps from other managers, it is possible for these other managers to harness some of the ruleset intelligence to aid with their event processing. In our lab, we tried this out with NetFinity. Since NetFinity does not have a correlation capability of its own, we decided to make use of the ruleset Pass on Match function to help process NetFinity traps.

In this example, we configured a ruleset to correlate traps from NetFinity and to generate a NetFinity alert if a certain sequence of traps was detected. Specifically, the ruleset was set up to detect if a high disk load trap from NetFinity was followed by traps that indicated low CPU and low adapter load. If such a sequence of traps was detected, the ruleset executed a remote command that generated a NetFinity alert in the originating NetFinity machine. Figure 121 illustrates the logical flow of traps in this example.

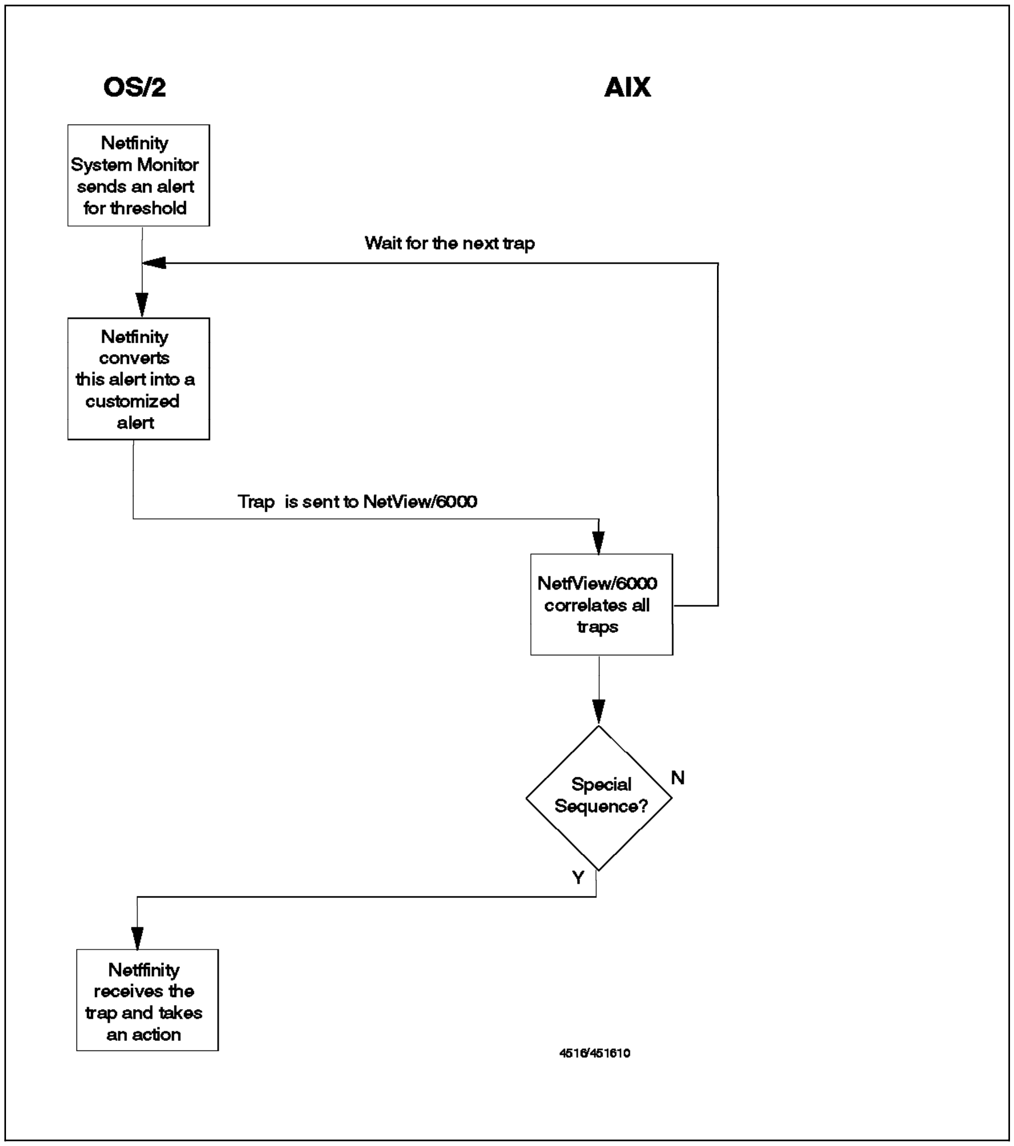

OS/2
AIX
Netfinity System Monitor sends an alert for threshold
Wait for the next trap
Netfinity converts this alert into a customized alert
Trap is sent to NetView/6000
NetfView/6000 correlates all traps
Special Sequence?
N
Y
Netffinity receives the trap and takes an action
4516/451610

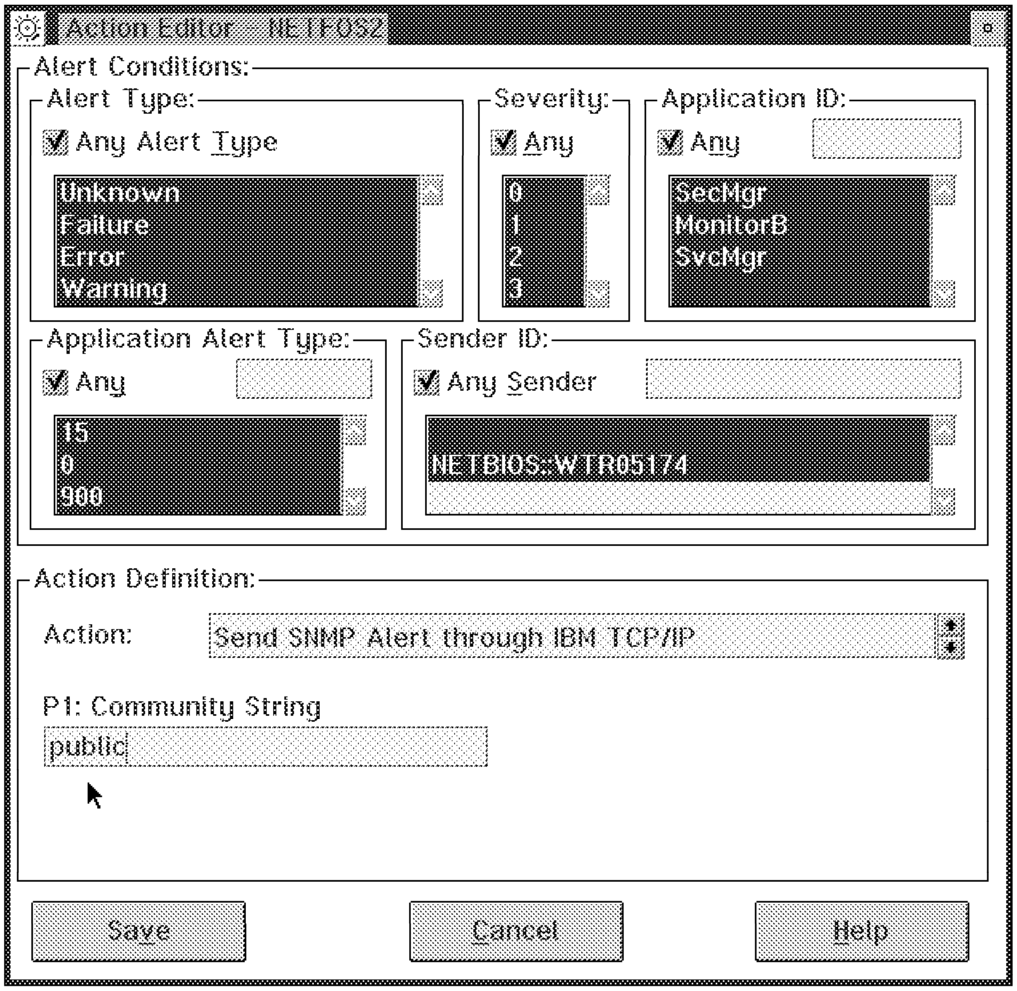
Figure 121. Diagram Showing Correlation of NetFinity Traps

6.2.10.1 Configuring the NetFinity Traps for this Example

NetFinity generates all its traps with a single specific trap ID, the actual meaning of the event being contained in a rather lengthy text string in a variable within the trap. The Event Attributes ruleset node can be used to check for the existence of a word at a given position within a trap parameter (see Figure 98 on page 128 for an example of this). Using this approach carries some risks, however, because the text format may vary with a new release of NetFinity.

Instead of using this feature, we chose in this case to configure the traps in NetFinity into a simplified version, to make the ruleset processing less complex. We configured the NetFinity alerts by completing the following steps:

1. Our first step was to be sure that NetFinity was forwarding its alerts to our target NetView for AIX manager. In order to do this, we configured TCP/IP on the NetFinity machine to include our NetView for AIX manager as a SNMP trap destination. We also configured NetFinity to forward all of its alerts over TCP/IP as shown in Figure 122.


Action Editor - NETFOS2

Alert Conditions:

Alert Type: ☑ Any Alert Type — Unknown, Failure, Error, Warning

Severity: ☑ Any — 0, 1, 2, 3

Application ID: ☑ Any — SecMgr, MonitorB, SvcMgr

Application Alert Type: ☑ Any — 15, 0, 900

Sender ID: ☑ Any Sender — NETBIOS::WTR05174

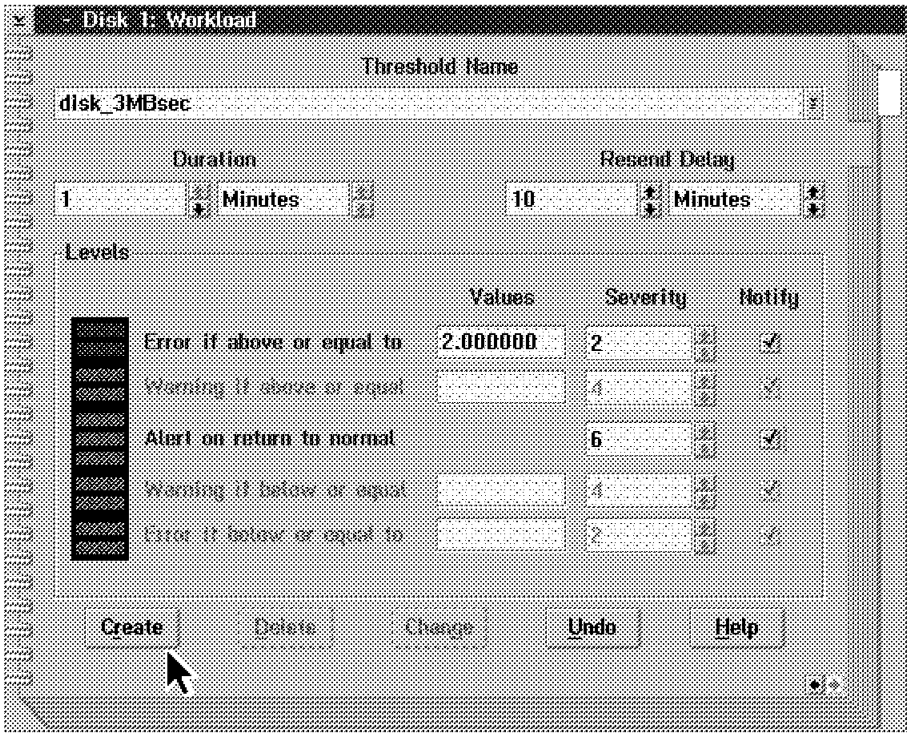
Action Definition:

Action: Send SNMP Alert through IBM TCP/IP

P1: Community String: public

Figure 122. Configuring NetFinity to Forward all Alerts through TCP/IP

2. We then defined the initial three NetFinity threshold alerts. These thresholds were defined to send alerts concerning high disk load, low CPU, and low adapter load. Figure 123 on page 153, Figure 124 on page 153, and Figure 125 on page 154 show how these alerts were defined.



Threshold Name: disk_3MBsec

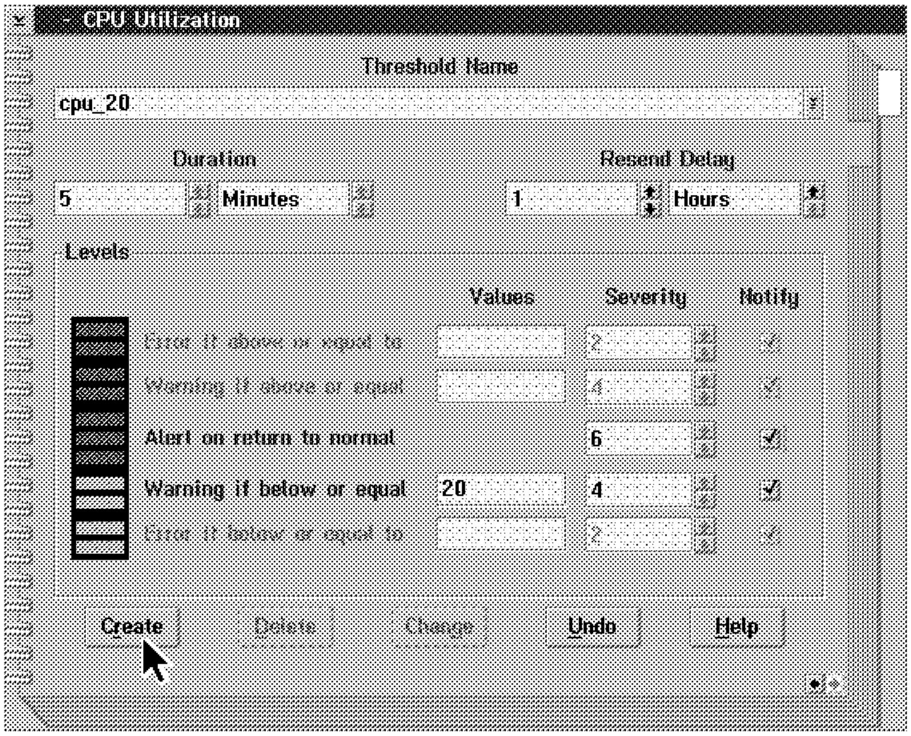
Duration: 1 Minutes

Resend Delay: 10 Minutes

Levels

	Values	Severity	Notify
Error if above or equal to	2.000000	2	✓
Warning if above or equal		4	✓
Alert on return to normal		6	✓
Warning if below or equal		4	✓
Error if below or equal to		2	✓

Figure 123. Defining the High Disk Load NetFinity Alert



Threshold Name: cpu_20

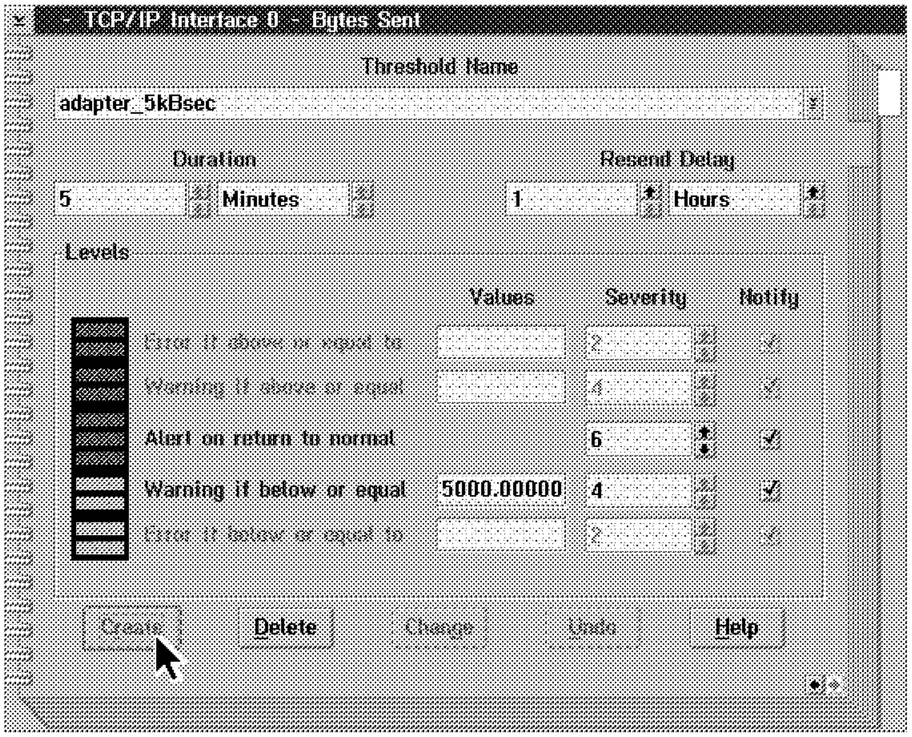
Duration: 5 Minutes

Resend Delay: 1 Hours

Levels

	Values	Severity	Notify
Error if above or equal to		2	✓
Warning if above or equal		4	✓
Alert on return to normal		6	✓
Warning if below or equal	20	4	✓
Error if below or equal to		2	✓

Figure 124. Defining the Low CPU NetFinity Alert



- TCP/IP Interface 0 - Bytes Sent

Threshold Name: adapter_5kBsec

Duration: 5 Minutes

Resend Delay: 1 Hours

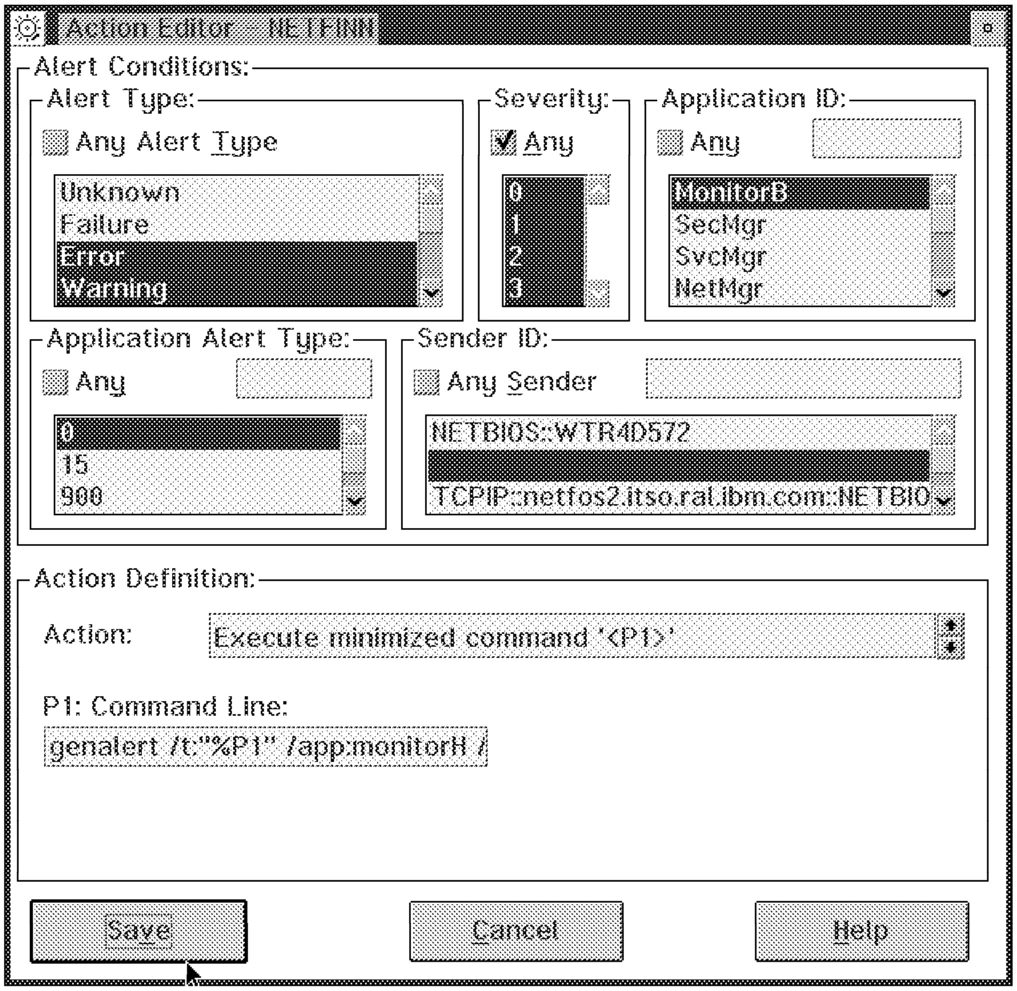
Levels

	Values	Severity	Notify
Error if above or equal to		2	✓
Warning if above or equal		4	✓
Alert on return to normal		6	✓
Warning if below or equal	5000.00000	4	✓
Error if below or equal to		2	✓

Figure 125. Defining the Low Adapter Load NetFinity Alert

3. Our next step was to convert the threshold traps into our customized alert. This was accomplished by configuring the Action Editor in NetFinity. The key to this step was to reduce the description of our customized alert to a single word. In this way, the ruleset could distinguish the trap without having to match individual words from the variable. In order to reduce the description, we simply made our description equal to the name of the threshold. Figure 126 on page 155 shows how we did this. Notice that we passed the threshold name to the customized trap via the parameter: %P1. The complete command needed to generate the alert in NetFinity is:

   ```
   genalert /t:"%P1"/app:monitorB/sev:3/type:dskwrn/atype:0011
   ```


Action Editor - NETFINH

Alert Conditions:

Alert Type:
▨ Any Alert Type
Unknown
Failure
Error
Warning

Severity:
✓ Any
0
1
2
3

Application ID:
▨ Any
MonitorB
SecMgr
SvcMgr
NetMgr

Application Alert Type:
▨ Any
0
15
900

Sender ID:
▨ Any Sender
NETBIOS::WTR4D572
TCPIP::netfos2.itso.ral.ibm.com::NETBIO

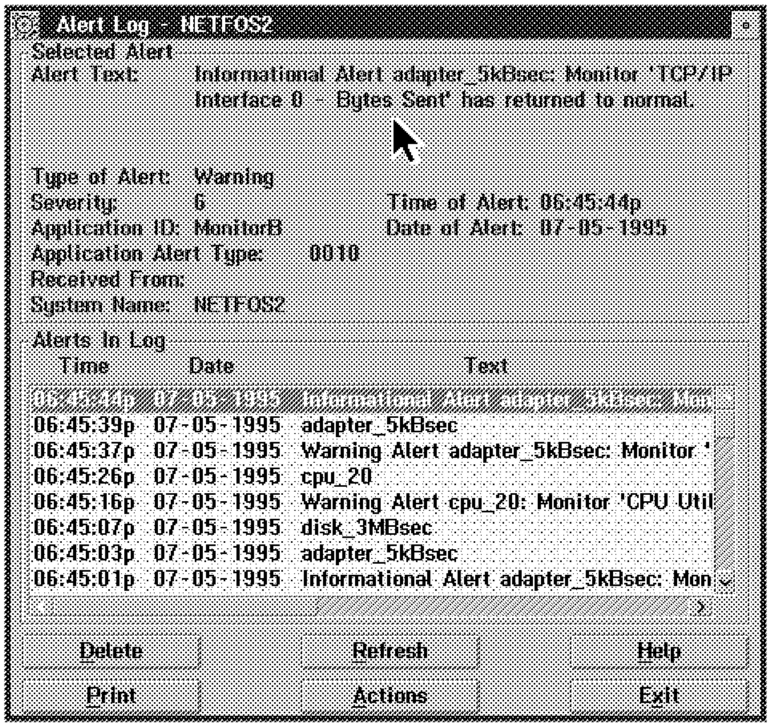
Action Definition:

Action: Execute minimized command '<P1>'

P1: Command Line:
genalert /t:"%P1" /app:monitorH /

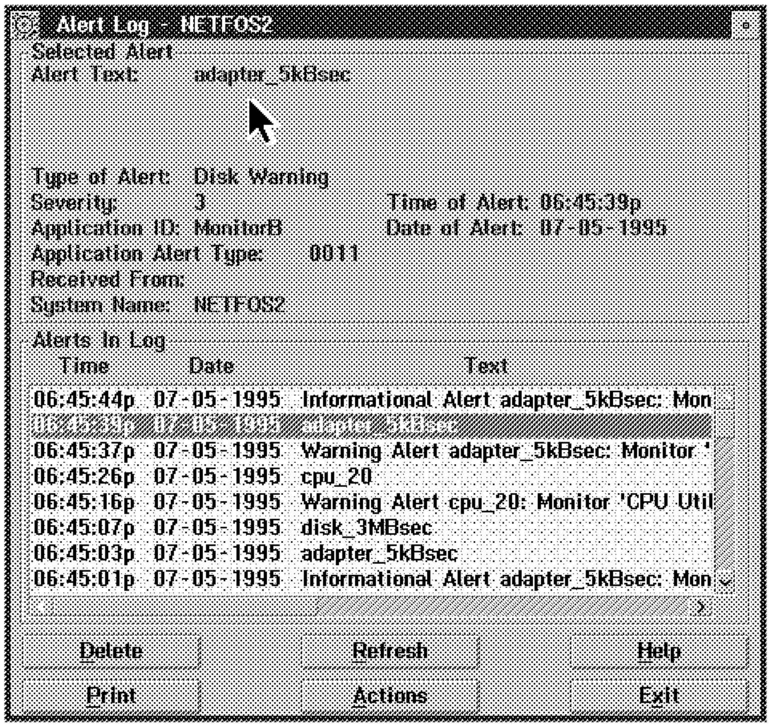
Figure 126. Converting an Alert by Generating a New One

Our next two figures show the results of converting an alert. Figure 127 on page 156 shows the standard threshold alert as it appears in the Alert log, and Figure 128 on page 156 shows the converted, customized alert.



```
Alert Log - NETFOS2
Selected Alert
Alert Text:      Informational Alert adapter_5kBsec: Monitor 'TCP/IP
                 Interface 0 - Bytes Sent' has returned to normal.

Type of Alert:   Warning
Severity:        6                  Time of Alert: 06:45:44p
Application ID:  MonitorB           Date of Alert: 07-05-1995
Application Alert Type:    0010
Received From:
System Name:     NETFOS2
Alerts In Log
  Time        Date                    Text
06:45:44p  07-05-1995  Informational Alert adapter_5kBsec: Mon
06:45:39p  07-05-1995  adapter_5kBsec
06:45:37p  07-05-1995  Warning Alert adapter_5kBsec: Monitor '
06:45:26p  07-05-1995  cpu_20
06:45:16p  07-05-1995  Warning Alert cpu_20: Monitor 'CPU Util
06:45:07p  07-05-1995  disk_3MBsec
06:45:03p  07-05-1995  adapter_5kBsec
06:45:01p  07-05-1995  Informational Alert adapter_5kBsec: Mon
```

Figure 127. An Alert before Conversion



```
Alert Log - NETFOS2
Selected Alert
Alert Text:      adapter_5kBsec

Type of Alert:   Disk Warning
Severity:        3                  Time of Alert: 06:45:39p
Application ID:  MonitorB           Date of Alert: 07-05-1995
Application Alert Type:    0011
Received From:
System Name:     NETFOS2
Alerts In Log
  Time        Date                    Text
06:45:44p  07-05-1995  Informational Alert adapter_5kBsec: Mon
06:45:39p  07-05-1995  adapter_5kBsec
06:45:37p  07-05-1995  Warning Alert adapter_5kBsec: Monitor '
06:45:26p  07-05-1995  cpu_20
06:45:16p  07-05-1995  Warning Alert cpu_20: Monitor 'CPU Util
06:45:07p  07-05-1995  disk_3MBsec
06:45:03p  07-05-1995  adapter_5kBsec
06:45:01p  07-05-1995  Informational Alert adapter_5kBsec: Mon
```

Figure 128. An Alert after Conversion

6.2.10.2 Creating the Ruleset

The ruleset for this example was configured to detect if a certain sequence of NetFinity traps were forwarded to NetView for AIX. The completed ruleset is shown in Figure 129.

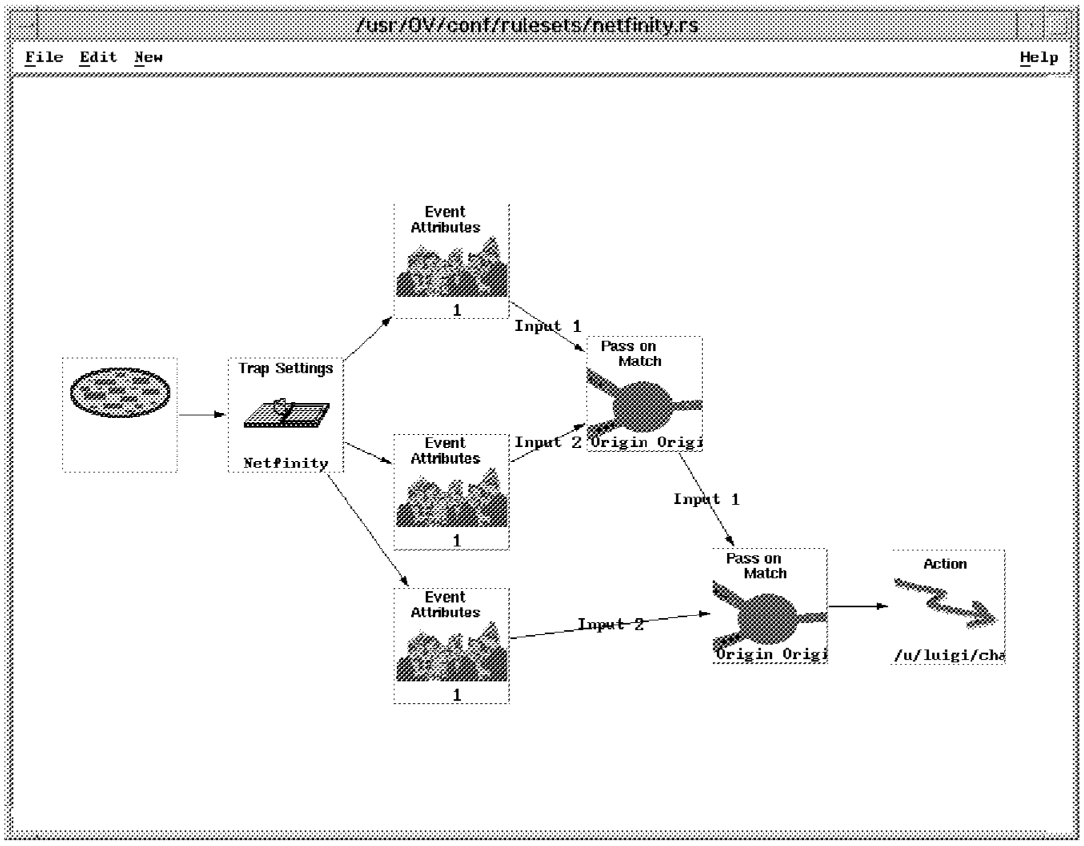
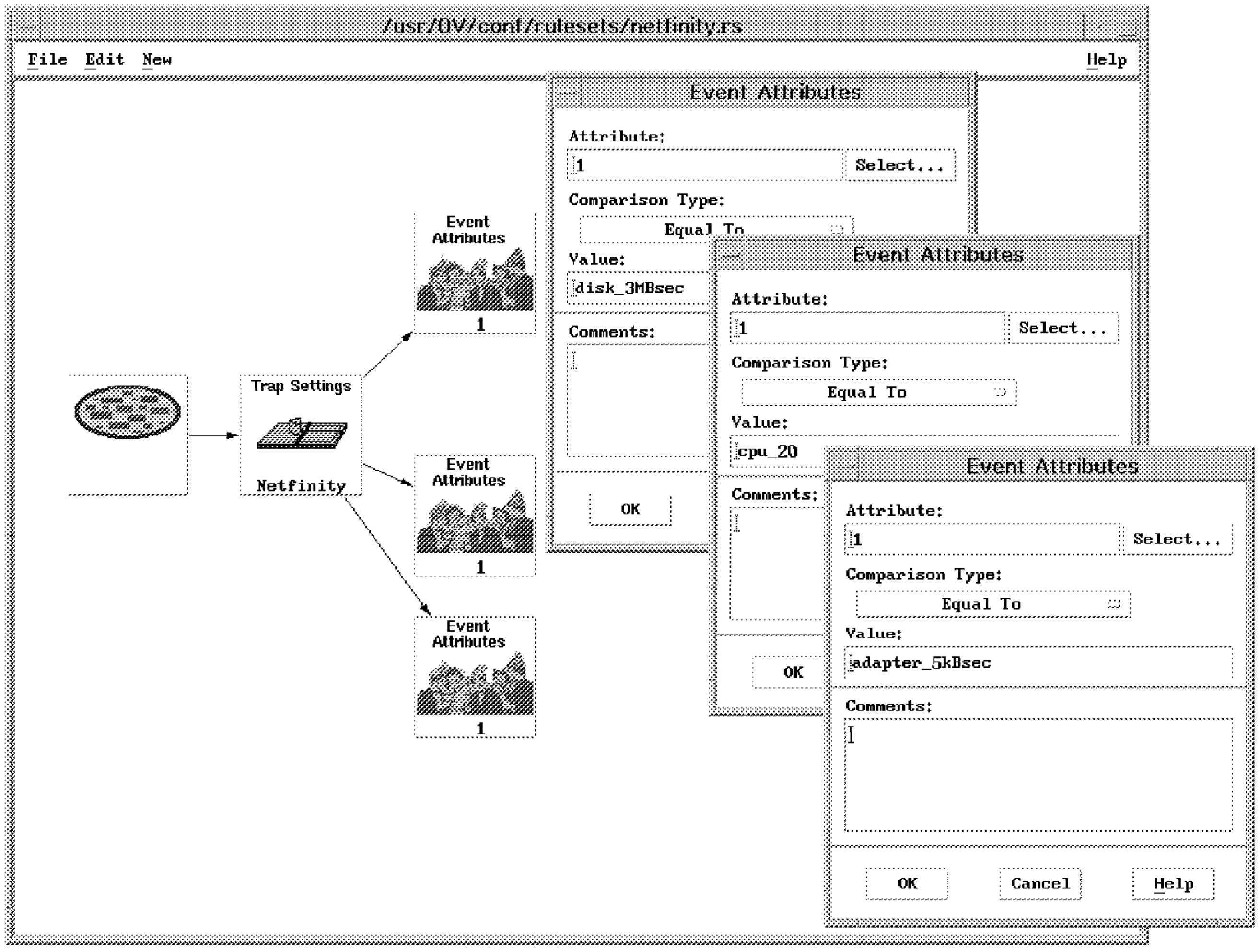


Figure 129. Completed Ruleset for Supplementing Other Managers

This ruleset was constructed by completing the following steps:

1. We first identified each of the three converted NetFinity traps via the Events Attributes node. The key here was to determine if the first attribute in the trap was equal to the name of the threshold originally defined in Netifinity. Figure 130 on page 158 shows how the traps were identified.



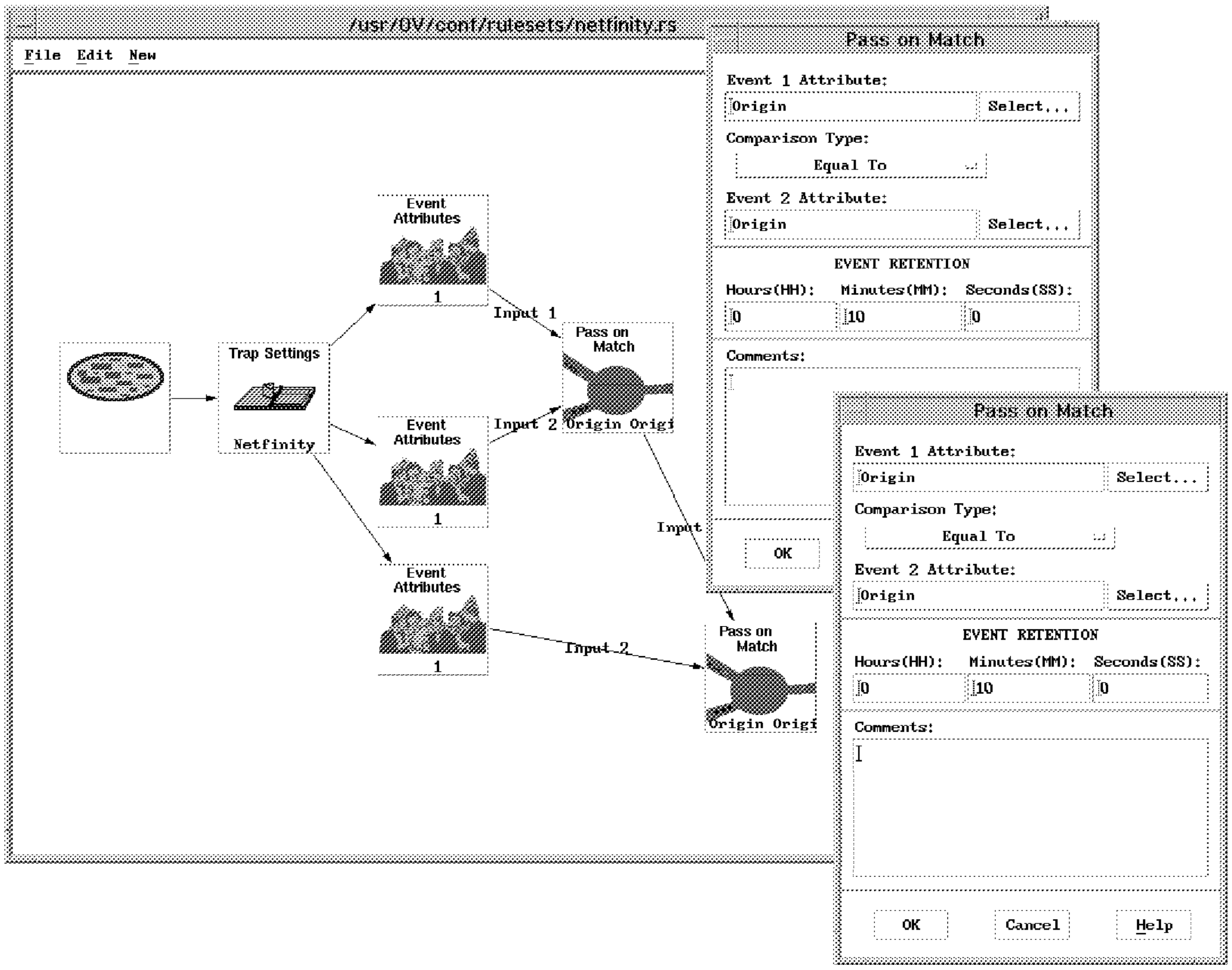
Attribute: 1
Comparison Type: Equal To
Value: disk_3MBsec
Comments:

Attribute: 1
Comparison Type: Equal To
Value: cpu_20
Comments:

Attribute: 1
Comparison Type: Equal To
Value: adapter_5kBsec
Comments:

Figure 130. Adding the Event Attributes Nodes

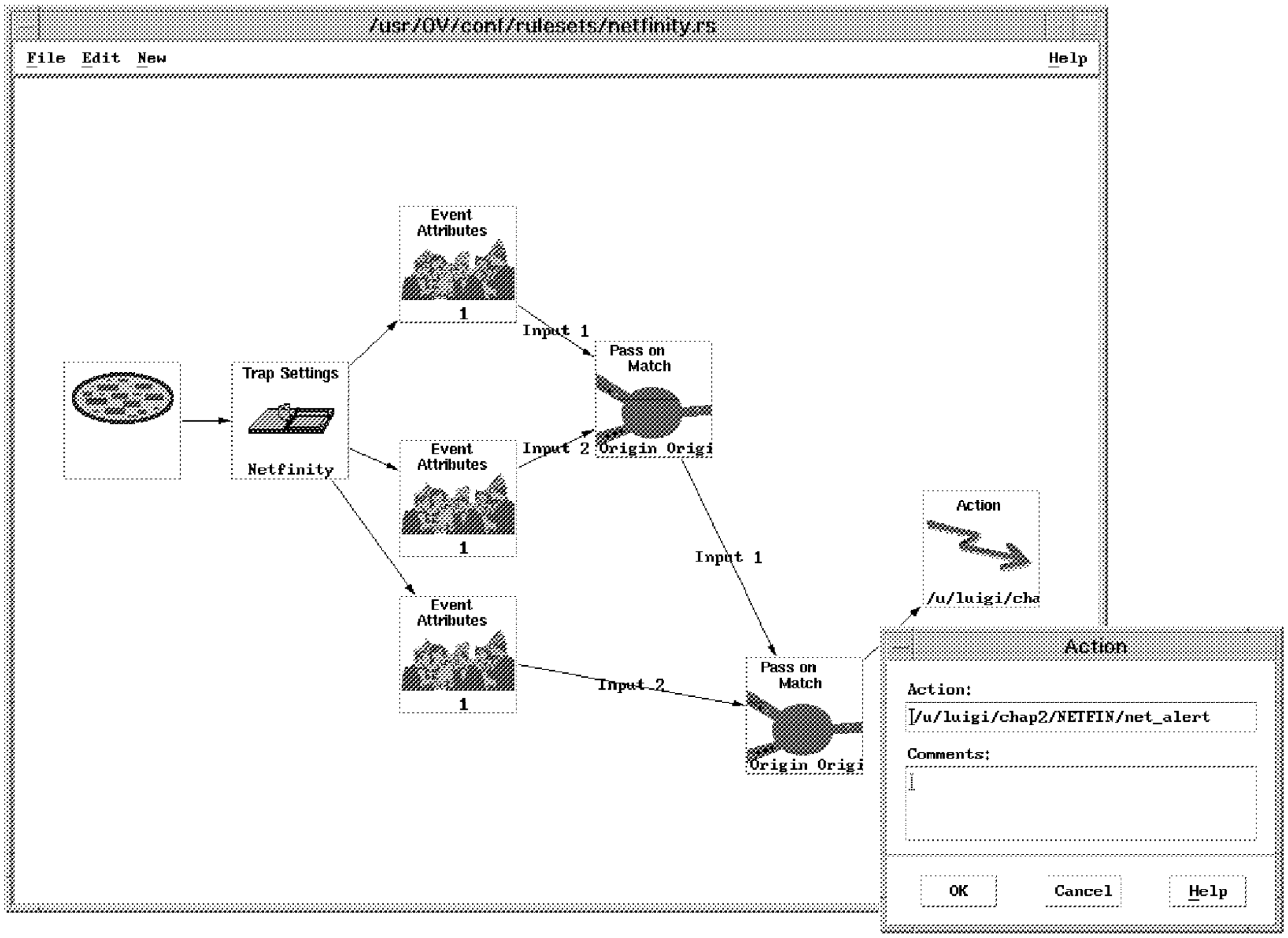
2. We then specified the specific sequence of traps by using the Pass on Match node. Figure 131 on page 159 shows how these nodes were configured. Note that we first correlated the High Disk Load trap with the Low CPU trap. If the Disk Load trap was followed by the CPU trap within ten minutes, the Disk Load trap was passed to the second correlation node. This second correlation node determined if the newly passed Disk Load trap was now followed by a Low Adapter Load trap.



Event 1 Attribute: Origin (Select...)
Comparison Type: Equal To
Event 2 Attribute: Origin (Select...)
EVENT RETENTION
Hours(HH): 0 Minutes(MM): 10 Seconds(SS): 0
Comments:

Figure 131. Adding the Correlation Attributes Nodes

3. If the Disk Load trap was passed by both correlation nodes, we configured an action to take place as shown in Figure 132 on page 160. The defined action was to execute a shell script that started a command on the NetFinity machine via TCP/IP's REXEC. This script is shown in Figure 133 on page 160.



Action:
/u/luigi/chap2/NETFIN/net_alert

Comments:

Figure 132. Adding the Action Node

```
#!/bin/ksh
HOME=/u/luigi
COMMAND=″genalert /t:″The disk load doesn′t match the general workload. \
        Configuration problems !!!″ /sev:3 /app:NetAix /type:SYSWARN    \
        /atype:0000a″
rexec netfinn.itso.ral.ibm.com $COMMAND
```

Figure 133. Net_alert Shell

Note that it was very important to define the variable HOME inside the shell, because rexec looked for the file .netrc in the home directory to determine the user ID and password for access to the remote machine. Our .netrc file was as shown in Figure 134

```
machine richard.itso.ral.ibm.com login richard password richard
machine netfinn.itso.ral.ibm.com login herbraun password herbraun
machine 9.24.104.114 login herbraun password herbraun
```

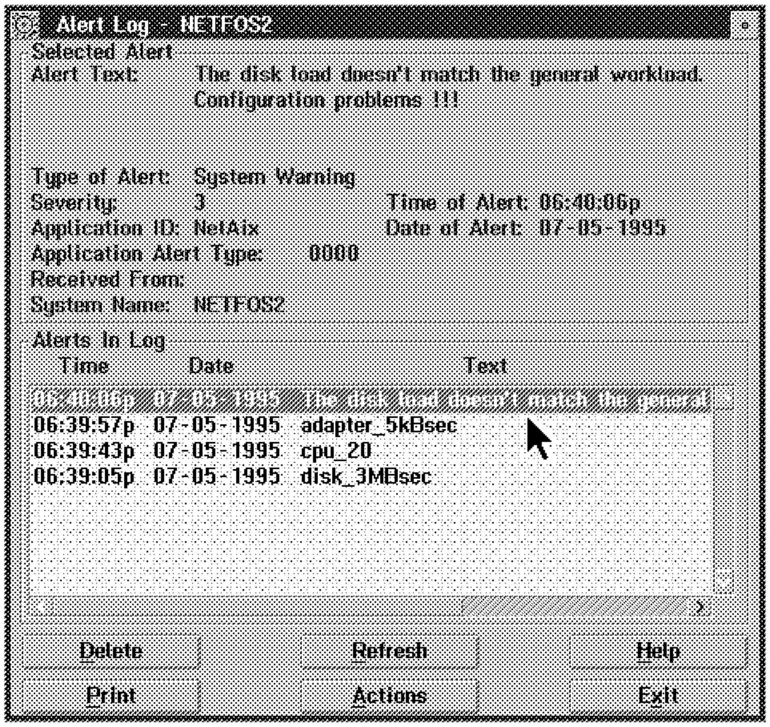
Figure 134. The .netrc File

6.2.10.3 Testing the Example

We tested the example in two ways. First we determined if the ruleset worked by generating the traps at our NetView for AIX workstation. The commands to generate these traps are listed below:

```
snmptrap rs600013.itso.ral.ibm.com .1.3.6.1.4.1.2.6.71.1.0 netfos2.itso.ral.ibm.com 6 9 0
        1 Octetstring "disk_3MBsec" 2 Octetstring netfos2.itso.ral.ibm.com
        3 Integer 3 4 Octetstring 04:16:00p 5 Octetstring 06-14-1995
        6 Octetstring Application 7 OctetString MonCritF 8 I nteger 0
        9 Octetstring netfos2.itso.ral.ibm.com

snmptrap rs600013.itso.ral.ibm.com .1.3.6.1.4.1.2.6.71.1.0 netfos2.itso.ral.ibm.com 6 9 0
        1 Octetstring "cpu_30" 2 Octetstring netfos2.itso.ral.ibm.com
        3 Integer 3 4 Octetstring 04:16:00p 5 Octetstring 06-14-1995
        6 Octetstring Application 7 OctetString MonCritF 8 Integer 0
        9 Octetstring netfos2.itso.ral.ibm.com

snmptrap rs600013.itso.ral.ibm.com .1.3.6.1.4.1.2.6.71.1.0 netfos2.itso.ral.ibm.com 6 9 0
        1 Octetstring "adapter_5kBsec" 2 Octetstring netfos2.itso.ral.ibm.com
        3 Integer 3 4 Octetstring 04:16:00p 5 Octetstring 06-14-1995
        6 Octetstring Application 7 OctetString MonCritF 8 Integer 0
        9 Octetstring netfos2.itso.ral.ibm.com
```

We also worked with the NetFinity machine in order to have the various threshold conditions arise. We were pleased to see that the ruleset was working and able to aid NetFinity in its correlation of alerts. Note that one of the alerts that was remotely executed from NetView for AIX is shown in Figure 135.


Alert Log - NETFOS2

Selected Alert

Alert Text: The disk load doesn't match the general workload. Configuration problems !!!

Type of Alert: System Warning
Severity: 3 Time of Alert: 06:40:06p
Application ID: NetAix Date of Alert: 07-05-1995
Application Alert Type: 0000
Received From:
System Name: NETFOS2

Alerts In Log

```
Time       Date        Text
06:40:06p  07-05-1995  The disk load doesn't match the general
06:39:57p  07-05-1995  adapter_5kBsec
06:39:43p  07-05-1995  cpu_20
06:39:05p  07-05-1995  disk_3MBsec
```

Figure 135. Remotely Executed Alert Received at NetFinity

6.2.11 Suppressing Events for Interfaces That Are Administratively Down

Often, IP routers are configured with interfaces that are for backup purposes. These interfaces will normally be down, unless a failure of some other component calls for them to be used. MIB-II provides a way of indicating interfaces that are configured but have been deliberately inactivated, by means of the two following objects in the Interfaces Table part of the MIB:

- ifOperStatus shows the operational status of the interface; whether it is active to the network or not.
- ifAdminStatus shows the administrative status of the interface; whether it is *intended* to be active or not.

The backup interface we described above should have both MIB instances set to Down.

Unfortunately, NetView for AIX does not take the ifAdminStatus field into account when it sets object status and sends Interface Down events. In this example we show a ruleset that partially alleviates this, by suppressing Interface down events if the interface is administratively down.

Figure 136 shows a flow diagram of the example.

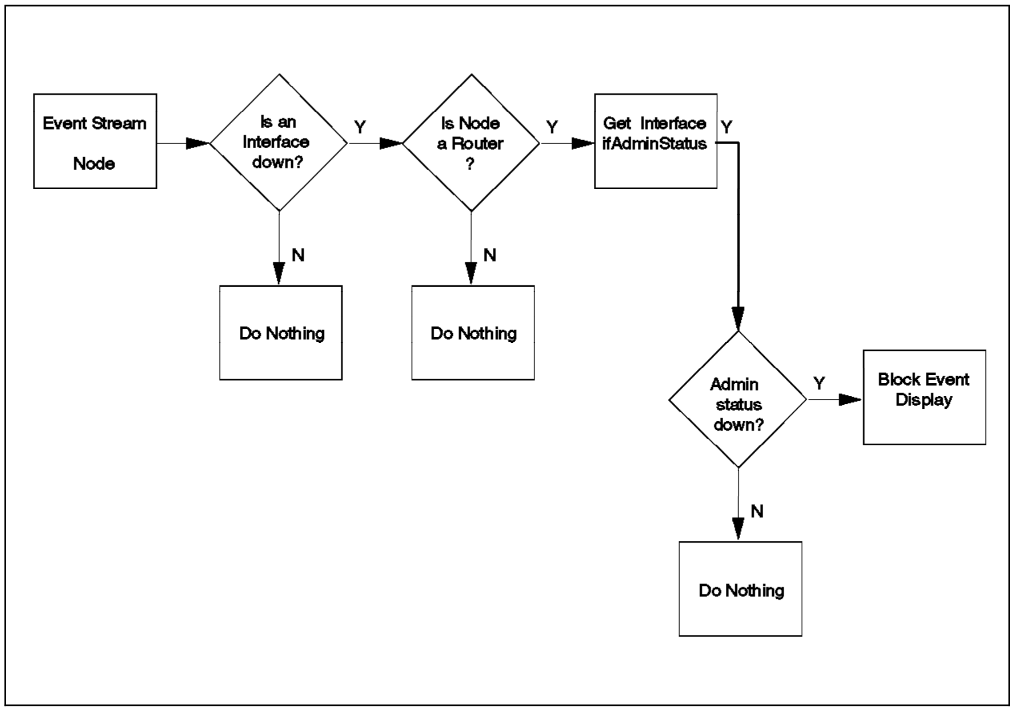


Figure 136. Flow Diagram for Check Administrative Status Example

The ruleset that we constructed to implement the example is shown in Figure 137 on page 163.

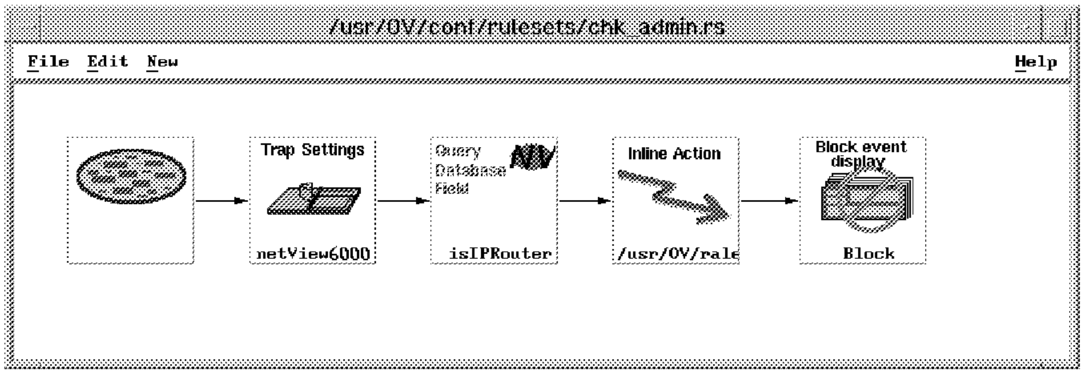


Figure 137. Completed Ruleset for Check Administrative Status Example

We created this ruleset using the following steps.

1. First we set the default behavior for the ruleset by double-clicking on the Event Stream node. In this case we wanted to see all events, unless they are blocked by the ruleset, so we chose a default of Pass.

2. Next we added the Trap Settings node and configured it as follows:

Enterprise Name	netview6000
Event Name	IBM_NVIDWN_EV
Specific	Specific 58916867

3. The concept of an interface that is administratively down usually applies to routers, so we further filtered the event stream by using a Get Database Field node to check the isIPRouter flag in the object database. See 6.2.5, “Using Traps to Override Status Color and Severity” on page 118 for an example of how to configure this node.

4. Next, we wanted to find the value of ifAdminStatus for the interface in question, by doing an SNMP get. The Compare MIB Variable node will do an SNMP get request, but unfortunately in this case we did not have enough information to use it. We need to get the specific instance of ifAdminStatus for the interface that is being reported as down. This means that we need to know the node name and the index number of the failing interface. By contrast, the interface down trap contains the following information:

   - The node name (in variable 2)
   - A text string that includes the interface name (in variable 3)

   What we therefore chose to do, was to write a shell script and a small program that uses the interface name from variable 3 and queries the NetView for AIX object database to determine the interface index number that it corresponds to. We then used the `snmpget` command to retrieve the value of ifAdminStatus. Figure 138 on page 164 shows the shell script and Figure 139 on page 164 shows the program that it calls to return the index number.

```
#!/bin/ksh

# Shell script that receives a node name and interface down trap text as
# input and then does an SNMP GET to check the ifAdminStatus. If it is
# down, the script exits with RC99

if_number=/usr/OV/raleigh/get_ifnumber $*

if [[ $? != "0" ]]
then
 set /usr/OV/bin/snmpget $1 .1.3.6.1.2.1.2.2.1.7.$if_number
 if [[ $4 = "up" ]]
 then
  exit 0
 else
  exit 99
 fi
fi
exit 0
```

Figure 138. chk_admin_stat Shell Script

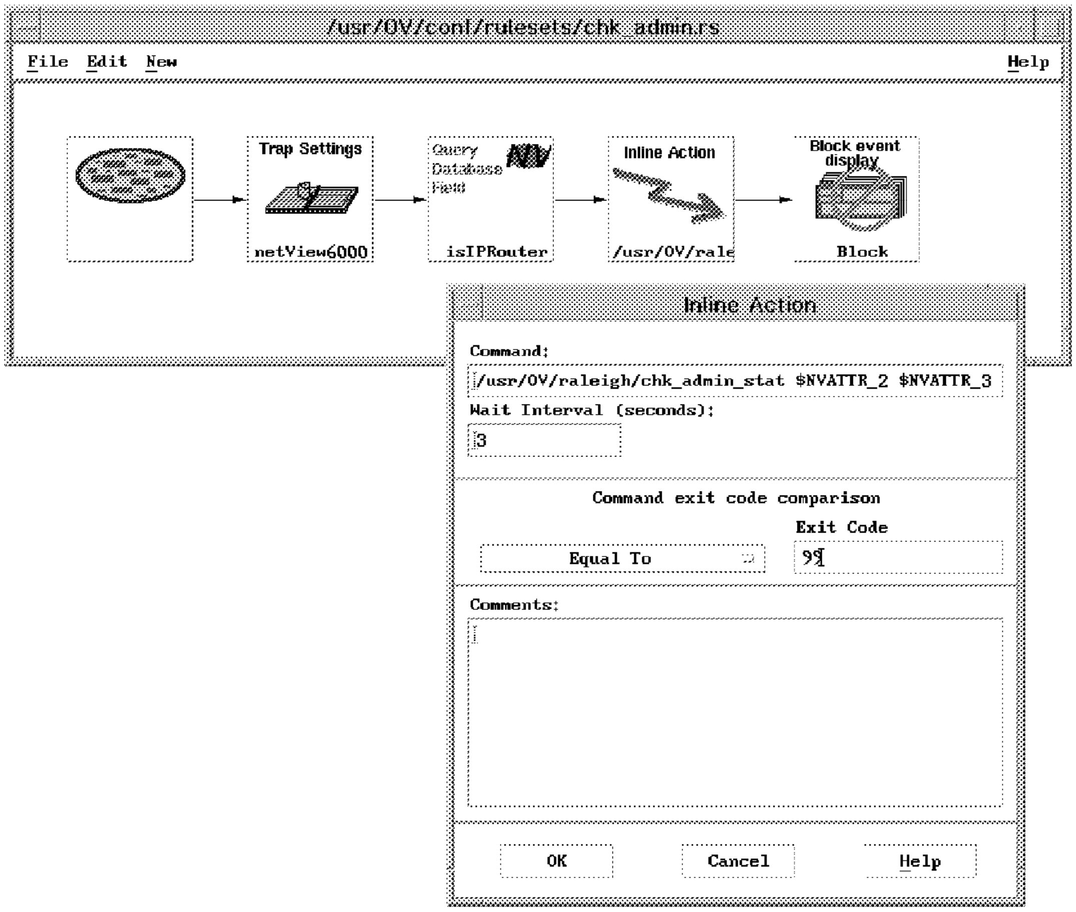
```
/*------------------------------------------------------------
get_ifnumber.c

This program looks for a specific interface name in the object
database record of a node and returns the index number (ie
relative position) of the interface
------------------------------------------------------------*/

#include <stdio.h>
#include <OV/ovw.h>

int main( int argc, char ** argv)
{
OVwObjectId  node_objid ;
OVwFieldValue   * iface_list ;
int             i ;
OVwListFieldEntry       * if_entry ;
char                    * if_label = "               " ;

if ( argc < 4 ) {
    printf("Usage: get_ifnumber node_name message_text\n") ;
    printf("Where node_name is the selection name for the node and\n") ;
    printf("message_text is the text of the interface down message\n") ;
     exit(0) ;
     }

/* Open the object database */
if (OVwDbInit() == EOF) {
     printf("OvwDbInit : %s\n", OVwErrorMsg(OVwError())) ;
     exit(0) ;
     }

/* Get the object ID of the node */
if ((node_objid = OVwDbSelectionNameToObjectId( argv[1] )) == NULL) {
   printf("OVwDbSelectionNameToObjectId : %s\n", OVwErrorMsg(OVwError())) ;
   exit(0) ;
   }
```

Figure 139 (Part 1 of 2). get_ifnumber.c Program to Extract Interface Index Number

```
/* Get the contents of the interface list field */
if ((iface_list = OVwDbGetFieldValue( node_objid,
                OVwDbFieldNameToFieldId( ″TopM Interface List″ ))) == NULL) {
   printf(″OVwDbGetFieldValue : %s\n″, OVwErrorMsg(OVwError())) ;
   exit (0) ;
   }

/* Read the entries in the list until we find a matching interface name */
if_entry = iface_list->un.list_val->list ;
    for (i=0; i < iface_list->un.list_val->count; i++ ) {
        sscanf(if_entry++->un.string_val, ″%s″, if_label) ;
        if ((strcmp(if_label, argv[3])) == NULL) {
           printf(″%d\n″, i+1) ;
           exit(i+1) ;
           }
        }

printf(″0\n″) ;
exit(0) ;
}
```

Figure 139 (Part 2 of 2). get_ifnumber.c Program to Extract Interface Index Number

5. Having created a command that will check the administrative status of the interface, we next needed to incorporate it into the ruleset. We have previously shown an example of using the Action node to invoke commands (6.2.4, “Automated Paging and E-Mail Notifications” on page 112). However, the Action node is executed after the ruleset has completed, by the actionsvr daemon. In this case we wanted the ruleset processing to be conditioned by the result of the command, so we used the Inline Action node instead. You can think of Inline Action as a kind of exit routine. Figure 140 on page 166 shows how we configured it in this case.



Command:
/usr/OV/raleigh/chk_admin_stat $NVATTR_2 $NVATTR_3
Wait Interval (seconds):
3

Command exit code comparison

Equal To

Exit Code
99

Comments:

Figure 140. Adding the Inline Action Node

Notice that an Inline Action is a *decision node*. It will pass an event only if the return code from the command matches the defined condition. In our case we are checking for a return code of 99 that was set by the *exit* statement within the shell script (see Figure 138 on page 164).

You should be careful when defining Inline Action nodes not to invoke a command which is long-running, since ruleset processing will be suspended while the action takes place. The node provides you with the Wait Interval field to limit the effect of long-running commands. If a command has not completed within the number of seconds specified it is abandoned. In this event, ruleset processing does not proceed to the next node.

6. Finally we added the Block Event Display node to prevent events that pass the Inline Action node from appearing in the event cards.

6.3 Combining ESE Rulesets

In most of the above examples we have tested the ruleset by creating a dynamic workspace and activating the ruleset in it. This is a reasonable approach to take; you aim to protect the user from a high volume of events by using filtering and prioritization.

If, however, this filtered event stream is directed to too many workspaces the benefits are lost. The corollary of this is that we want to have multiple rulesets active in a single workspace. Unfortunately nvevents does not support this possibility at present. The alternative is to combine all the rules you want to activate together into one monster ruleset. The disadvantage to this is that it becomes unwieldy to modify it using the ruleset editor.

The approach we recommend is to build ruleset fragments, like in our examples above and then to use the File→Include option to combine them into the complete ruleset.

6.3.1 Operation of the Include Function

As its name suggests, Include incorporates a second ruleset into the existing one. The best way to illustrate this is by way of an example. We combined the rulesets from 6.2.1, “Clearing Outstanding Events via Correlation” on page 92 and 6.2.11, “Suppressing Events for Interfaces That Are Administratively Down” on page 162 using the following steps:

1. Load ruleset correlation.rs (see Figure 66 on page 98) into the editor by selecting File→Open from the menu bar.
2. Include ruleset chk_admin.rs (see Figure 137 on page 163) into this ruleset by selecting File→Include.
3. Save the combined ruleset as combined.rs by selecting File→Save As.

The resulting ruleset is shown in Figure 141 on page 168.

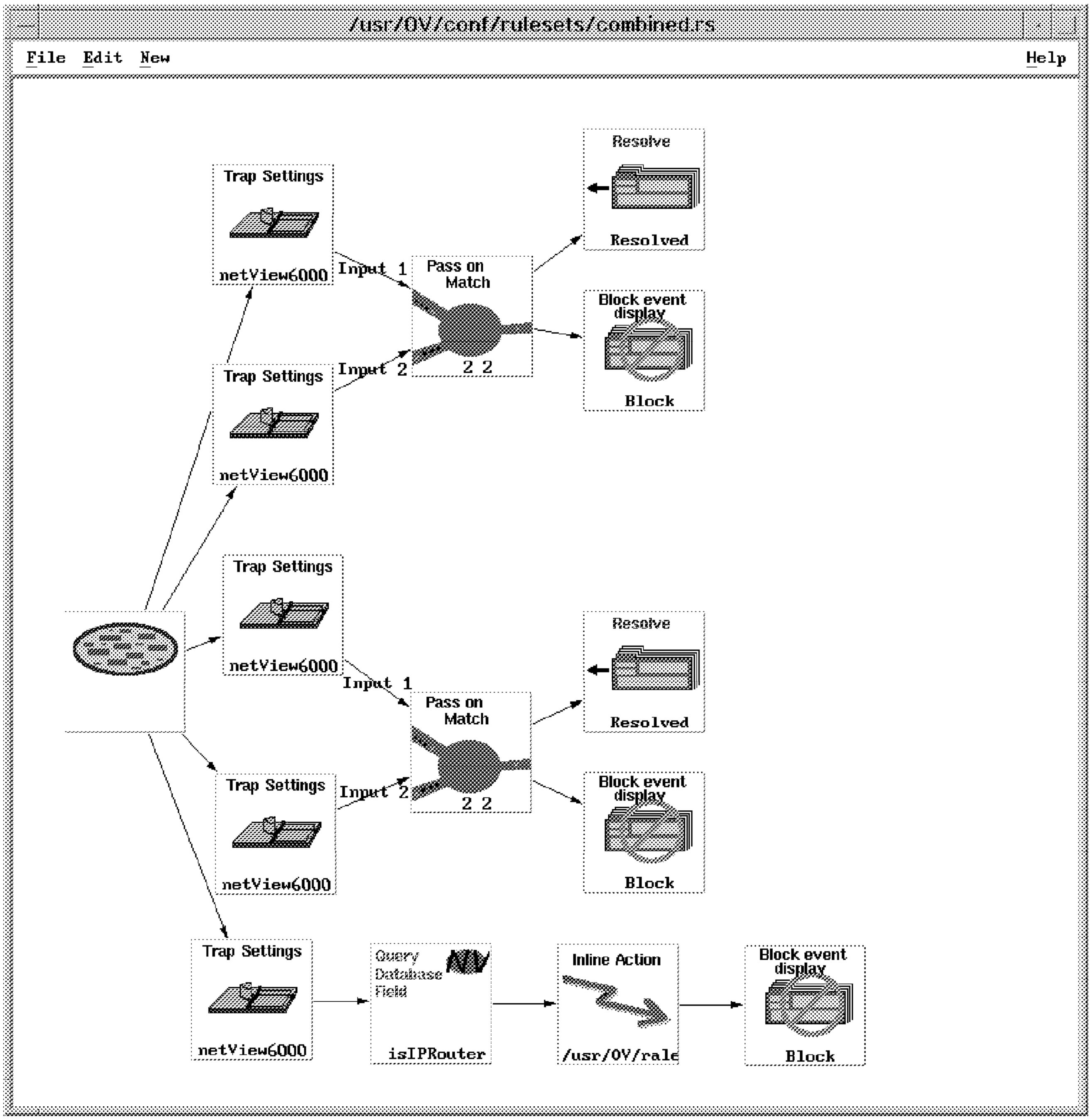


Figure 141. Combined Ruleset combined.rs

You can see that the Include function has taken a simplistic approach, merely adding the nodes from the second ruleset as an additional strand. It *could* have combined them further, for example both rulesets have a Trap Settings node that checks for Interface Down which could have become one in the combined ruleset. Because the ruleset processor tries to send each event along every path in the ruleset it does not matter that a decision node is duplicated, although it may be more efficient to combine them.

6.3.2 Conflicts when Combining Rulesets

When you use the Include function you need to keep in mind what function each ruleset performs, and make sure you are not introducing conflicts. For example, one ruleset may cause an event to be blocked and another may cause it to be forwarded. In general the rules for conflicts are:

- If you combine two rulesets with different default behaviors, the resulting ruleset will have the default behavior of the ruleset that you *insert into*, not the ruleset that you insert.
- If one path in the ruleset leads to a Block Event Display node and another leads to a Forward or Override node, the Block node will be ignored and Override will take priority over Forward. Note, however, that you will only ever get one copy of the event forwarded, even if it encounters multiple Forward or Override nodes.
- If two paths in the ruleset lead to Override nodes the ″most serious″ severity and/or status will take effect. So, for example, the ruleset shown in Figure 142 will cause the event to be displayed with severity Major and the node to change status to User2. ″Most serious″ in this case means in terms of the order of statuses shown on the list of options, which in order of increasing seriousness is: Unknown (blue), Normal (green), Marginal (yellow), Critical (red), Unmanaged (brown), Acknowledged (forest green), User1 (Pink), User2 (Purple).

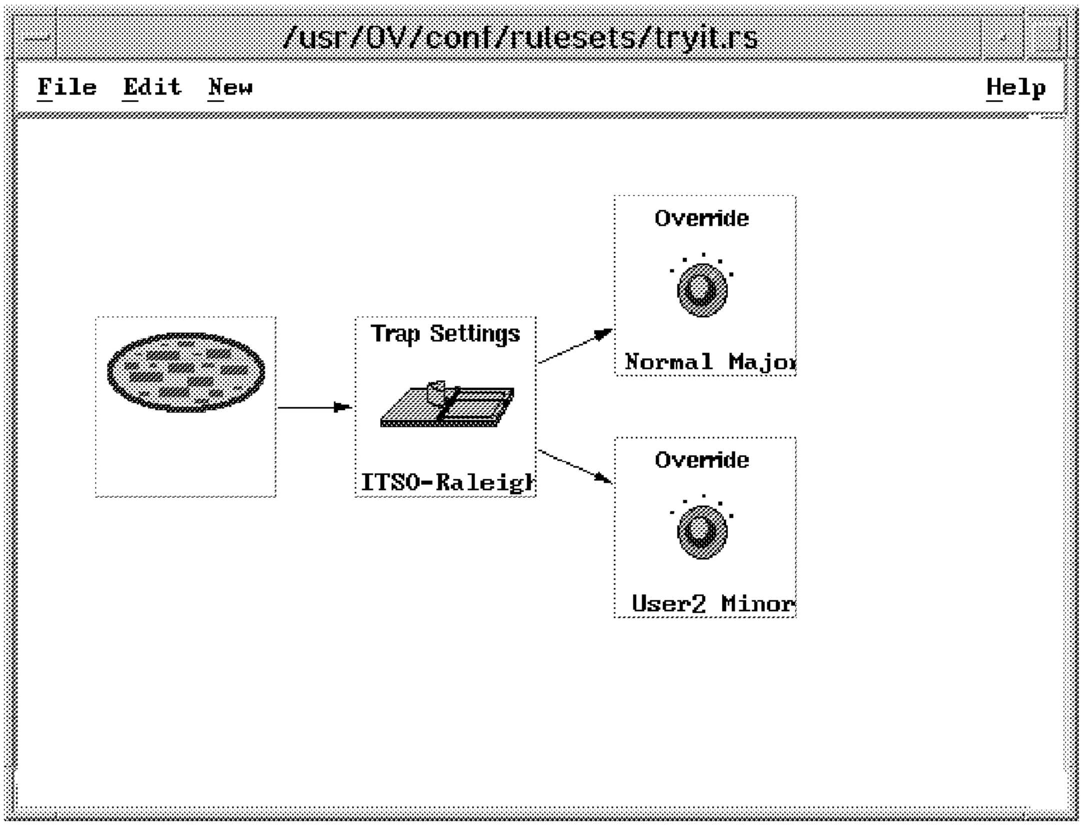


Figure 142. Example of Conflicting Override Nodes

6.4 Saving the Dynamic Workspace Environment

Now that you have the power of event rulesets at your disposal, you will probably want to always use filtered dynamic workspaces, with rulesets loaded into them. What you probably *don't* want to do is to have to set up the workspaces manually each time you restart your NetView for AIX EUI.

Fortunately, NetView for AIX V4R1M1 (the refresh level of the code) gives you the ability to save your configuration. To do this, you need to do the following:

1. Change the nvevents.loadEnvOnInit X-windows default to True. This resource is defined in file Nvevents in directory /usr/OV/app-defaults. You can change it globally there, or alternatively take a copy of the file and place it in your own home directory.
2. Start up the NetView for AIX EUI and create the dynamic events workspaces that you want.
3. Switch to the main nvevents display (the one labelled Events) by pressing the symbol on the left of the events display. Then select Options→Save Environment from the nvevents menu bar (note: this does not actually save anything, it just sets a flag).
4. Still in the main events display, select File→Exit from the nvevents menu bar. You will be prompted to confirm and will then receive a message that the environment has been saved in a new NvEnvironment subdirectory of your home directory.
5. Restart nvevents by selecting Monitor→Events→Current Events from the NetView for AIX main menu bar. You should see your dynamic event displays are restored.

Chapter 7. Using the Collection Facility with Event Rulesets

In Chapter 4, "The Collection Facility" on page 53 and Chapter 6, "Examples of NetView for AIX Rulesets" on page 85 we have seen examples that use two of the powerful new facilities of NetView for AIX Version 4, Collections and Rulesets. In this chapter, we discuss how we used the collection facility in combination with rulesets to help us manage events. We will describe the following examples:

- Receiving only events from a specific collection (7.1.1, "Receiving Only Events from One Collection").
- Overriding event severity on the basis of collection membership (7.1.2, "Manipulating Event Severity on the Basis of Node Importance" on page 175).
- Suppressing subsidiary events when a router is down (7.1.3, "Suppressing Subsidiary Events When a Router Is Down" on page 181).
- wtdepend. A package that provides a generalized way to suppress events and limit polling of resources that are dependent on an IP router interface (7.1.4, "The wtdepend Sample Application" on page 187).

7.1.1 Receiving Only Events from One Collection

In this first example, we wanted simply to set up a dynamic event workspace that only receives traps from a particular collection. At present there is no ruleset node that checks for membership of a given collection (at the time of writing it was planned as a future enhancement). However, we used the Inline Action node to execute the *wtcoll* sample code to do the job for us. The wtcoll program is described in 4.7, "The wtcoll Sample Program" on page 76 and the source code is listed in Appendix B, "C Code for the wtcoll Sample Program" on page 259.

7.1.1.1 Creating the Ruleset

The ruleset for this example was very simple. Figure 143 on page 172 shows the completed ruleset.

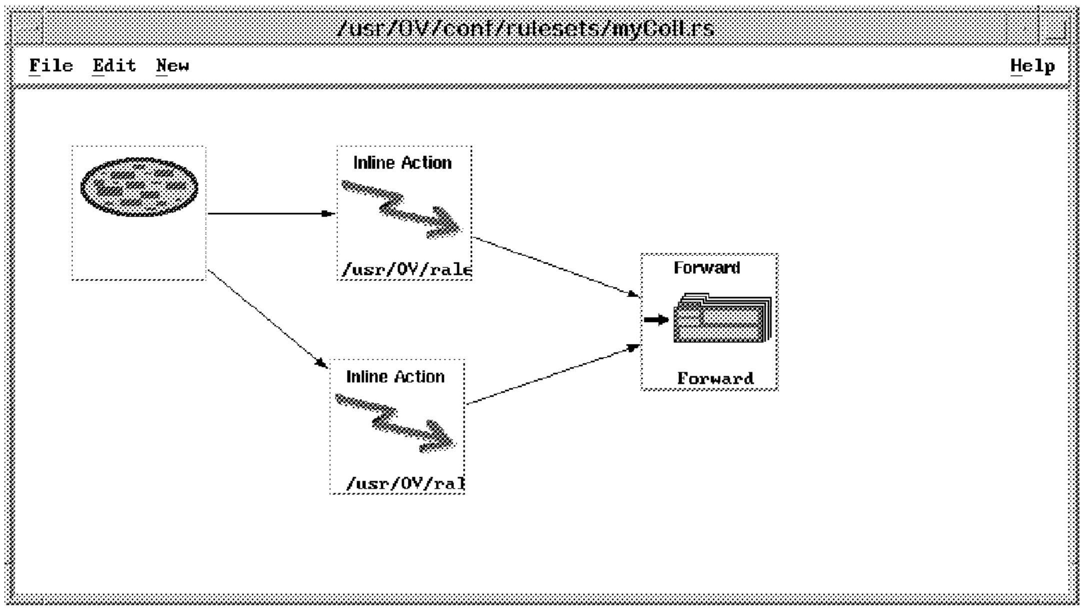
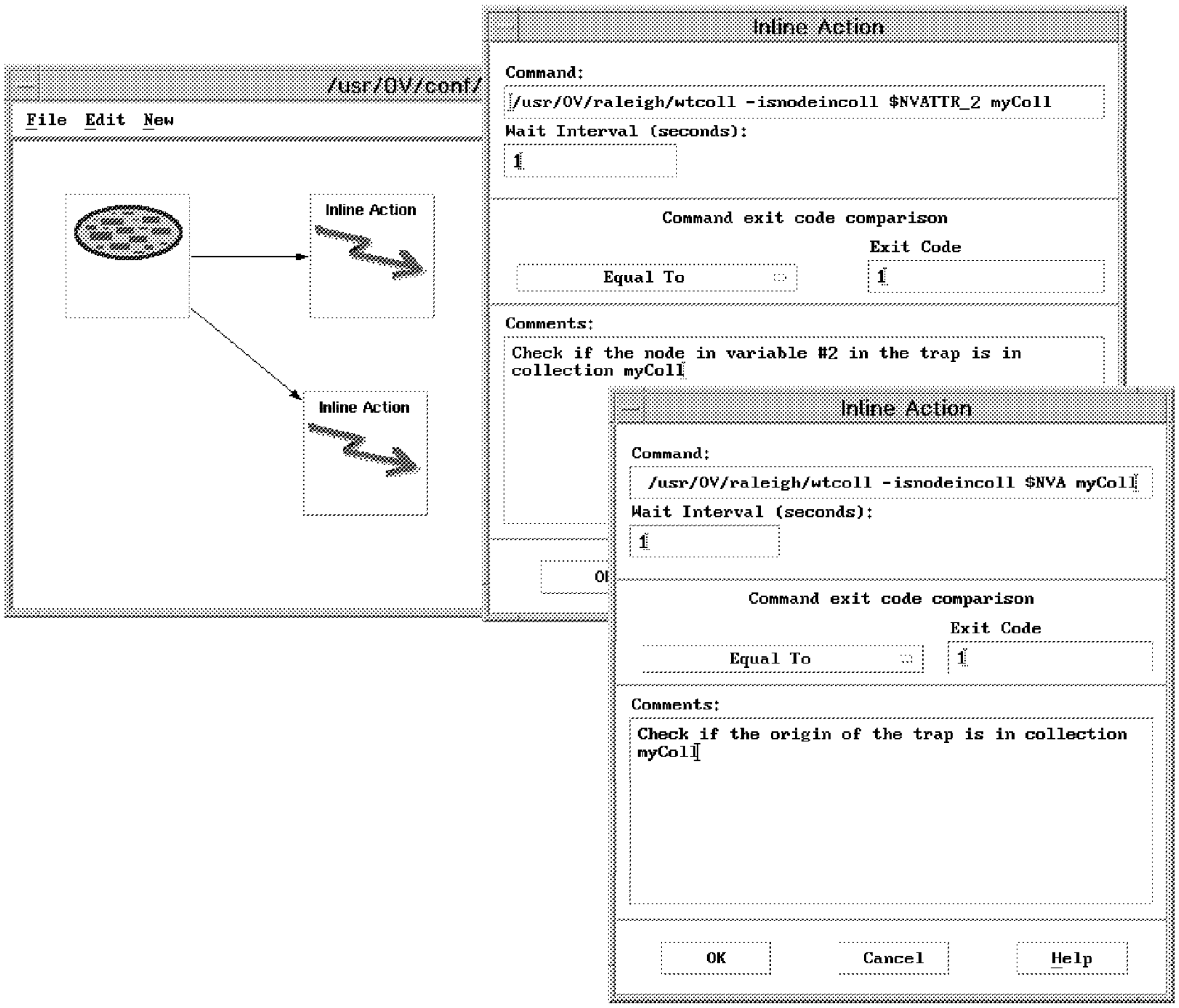


Figure 143. The Complete Ruleset. The default behavior was set to Block, so only events that pass the Inline Action decision nodes will appear.

You may wonder why there are two Inline Action nodes. The reason is that we want to show events that originate from the node itself *and* events reported by NetView for AIX or Systems Monitor for AIX. This means that we want one node that checks the origin of the trap and another that checks the second variable in the trap.

The Inline Action nodes were configured as shown in Figure 144 on page 173. It uses the -isnodeincoll option of wtcoll to check to see if the node is in collection myColl. wtcoll gives a return code of 1 if the node is in the collection, and 0 if it is not, or if an error occurs. Notice that we are using two of the available environment variables to substitute the second variable of the trap and the origin (agent address).



Command:
/usr/OV/raleigh/wtcoll -isnodeincoll $NVATTR_2 myColl
Wait Interval (seconds):
1
Command exit code comparison
Equal To
Exit Code
1
Comments:
Check if the node in variable #2 in the trap is in collection myColl

Command:
/usr/OV/raleigh/wtcoll -isnodeincoll $NVA myColl
Wait Interval (seconds):
1
Command exit code comparison
Equal To
Exit Code
1
Comments:
Check if the origin of the trap is in collection myColl

Figure 144. Adding the Inline Action Nodes

7.1.1.2 Testing the Ruleset

In order to test this example we completed the following steps:

1. First we created a collection called myColl. This was a simple node list collection (see 4.4.2, “Creating a Node List Collection” on page 61 for an example of how to do this). The resulting collection submap is shown in Figure 145 on page 174.

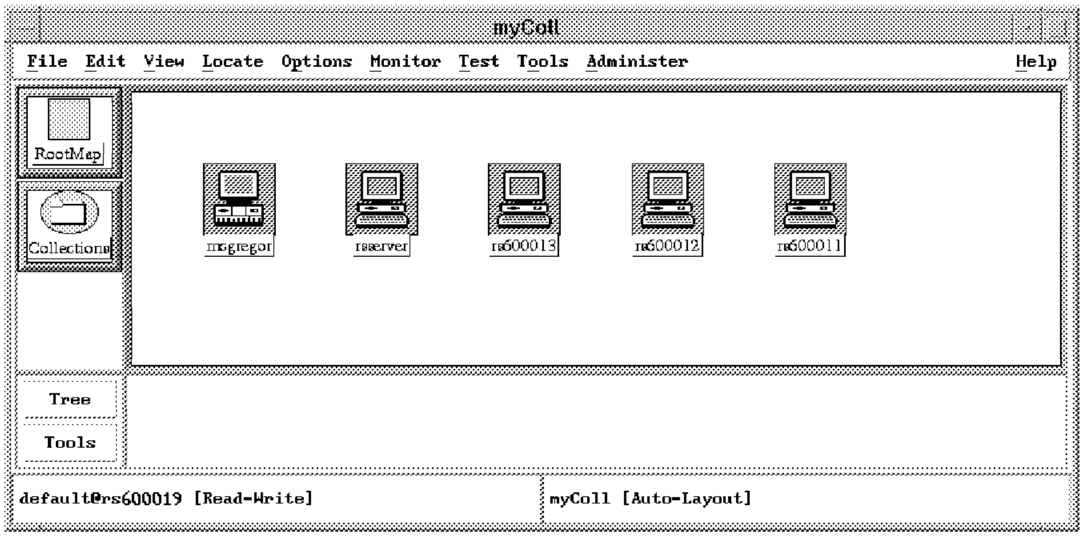

myColl
File Edit View Locate Options Monitor Test Tools Administer Help
RootMap
Collections
mcgregor
rsserver
rs600013
rs600012
rs600011
Tree
Tools
default@rs600019 [Read-Write]
myColl [Auto-Layout]

Figure 145. The myColl Collection

2. Next we opened a new dynamic workspace using the ruleset file that we created called myColl.rs.

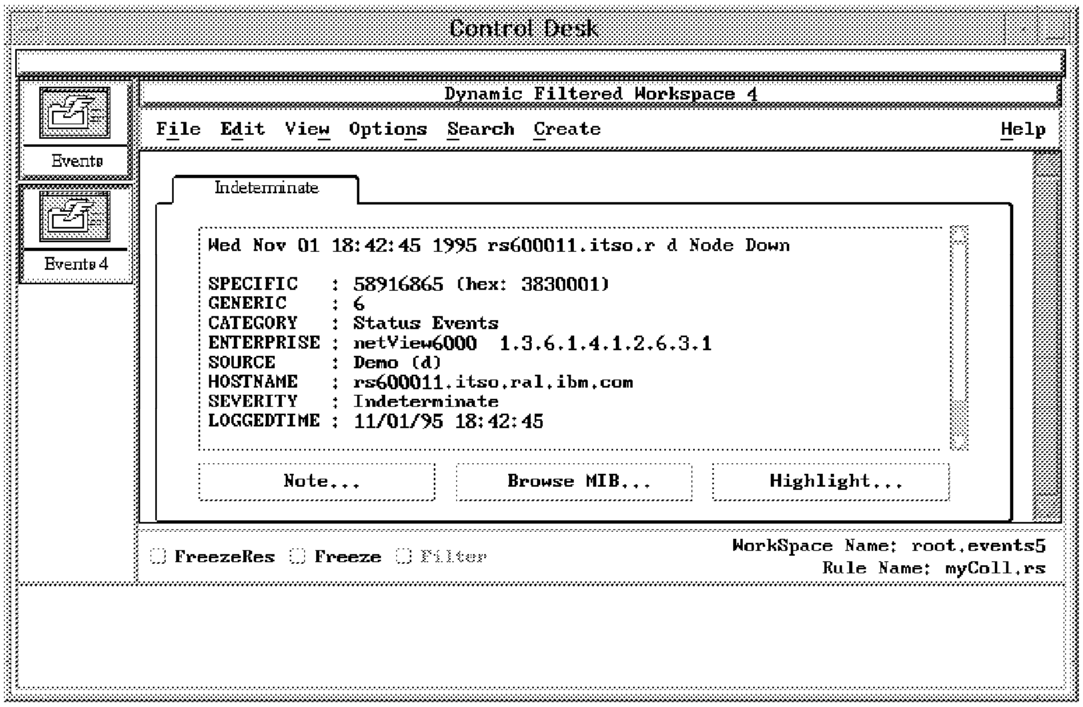
3. We then sent two events. One event for a member of the collection and one event for a node that is not a member. The trap commands that we used are as follows

   ```
   event -h rs600011.itso.ral.ibm.com -e NDWN_EV
   ```

   and

   ```
   event -h rs60003.itso.ral.ibm.com -e NDWN_EV
   ```

If our program and ruleset were correct, we should only receive the event from rs600011 since it is a member of our collection. This is the result we obtained as shown in Figure 146 on page 175.



```
Wed Nov 01 18:42:45 1995 rs600011.itso.r d Node Down

SPECIFIC   : 58916865 (hex: 3830001)
GENERIC    : 6
CATEGORY   : Status Events
ENTERPRISE : netView6000  1.3.6.1.4.1.2.6.3.1
SOURCE     : Demo (d)
HOSTNAME   : rs600011.itso.ral.ibm.com
SEVERITY   : Indeterminate
LOGGEDTIME : 11/01/95 18:42:45
```

```
WorkSpace Name: root.events5
Rule Name: myColl.rs
```

Figure 146. The Filtered myColl Workspace

7.1.2 Manipulating Event Severity on the Basis of Node Importance

A common problem in event management is being able to distinguish which events come from important nodes and which events come from nodes of lesser importance. One method that we used to overcome this problem involved manipulating the severity of traps. In this example, we changed the severity of traps on the basis of which collection the traps came from. If traps came from nodes that were part of a server collection, severity was upgraded to Major. If traps came from nodes that were part of a client collection, severity was downgraded to Minor. Figure 147 on page 176 illustrates the logical flow of this example.

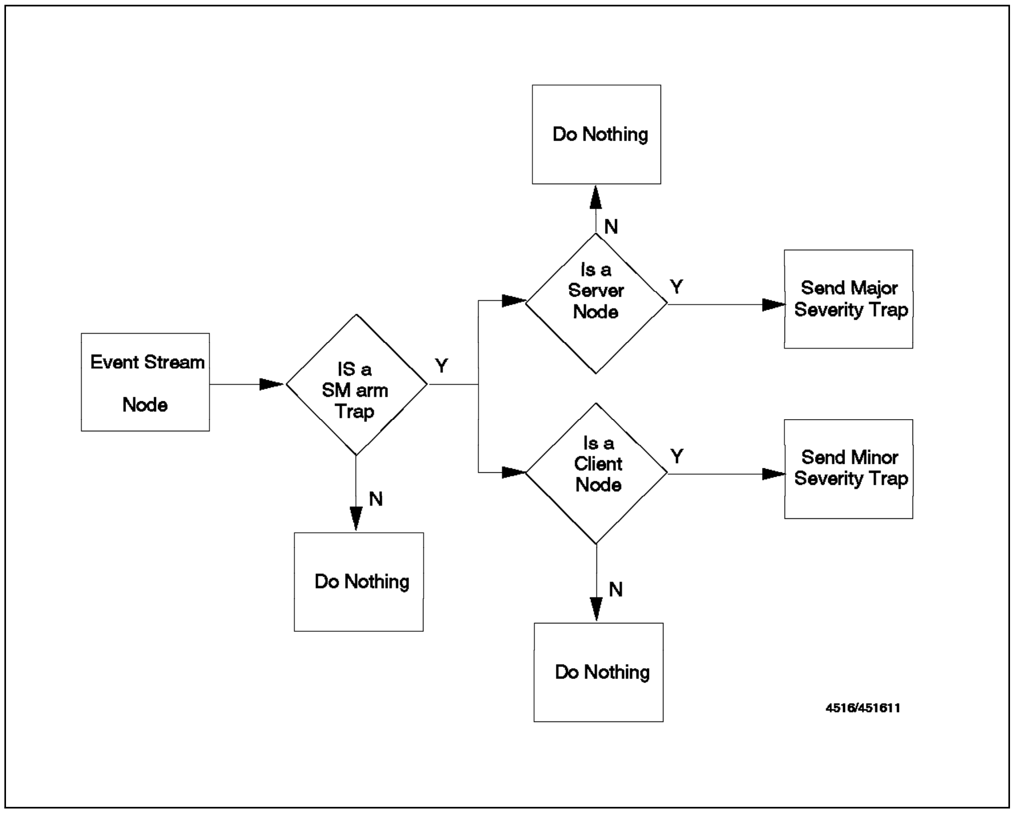


Figure 147. Flowchart of Manipulating Trap Severity Example

The processing for this example is very similar to the previous one. We again used the -isnodeincoll function of our wtcoll sample to check whether a node was part of a collection or not.

We first defined the server and client collections by using the Collection Editor. Figure 148 on page 177 and Figure 149 on page 178 show how this was done.

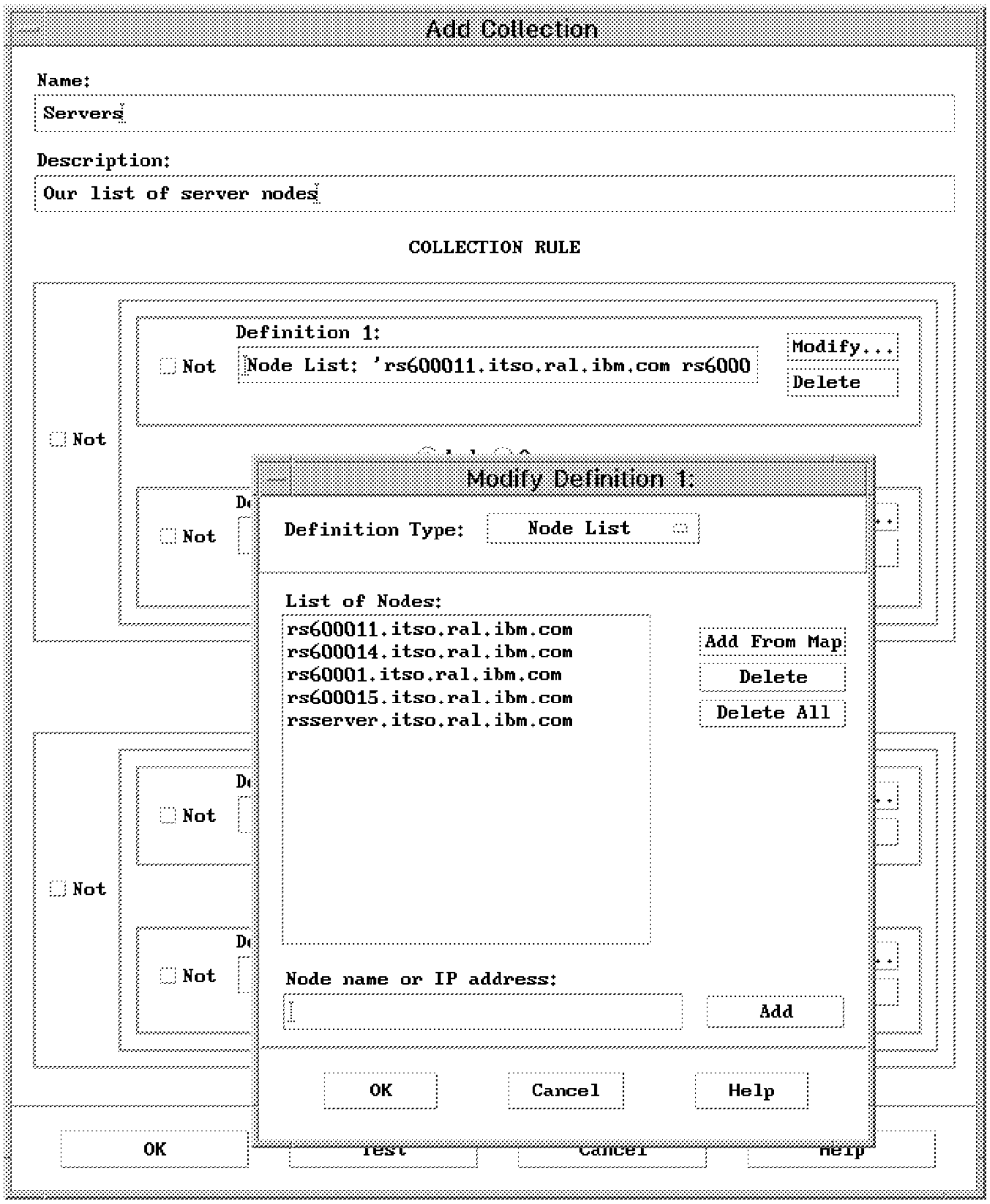


Figure 148. Creating the Servers Collection

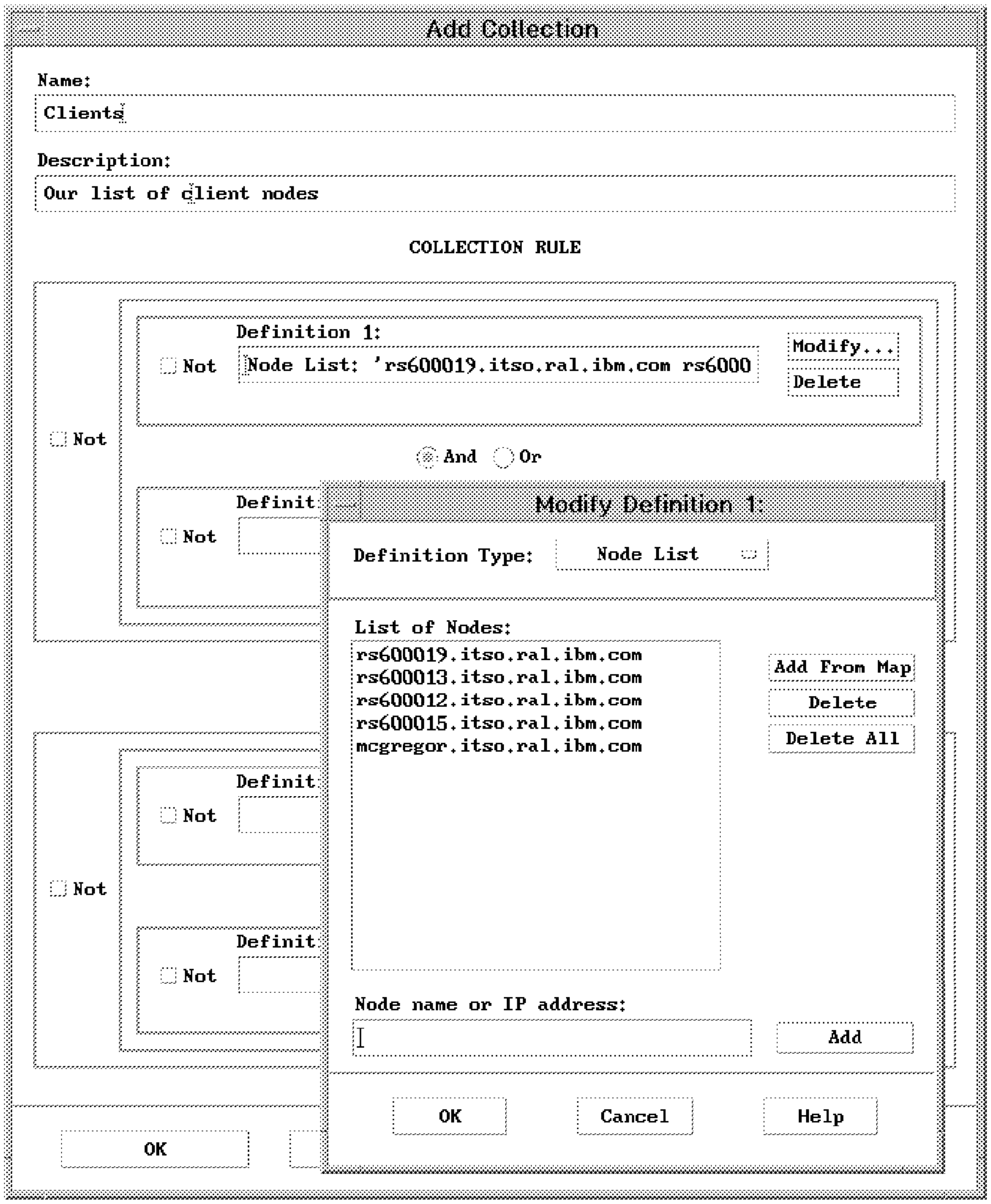


Figure 149. Creating the Clients Collection

7.1.2.1 Creating the Ruleset

The complete ruleset for this example is shown in Figure 150 on page 179.

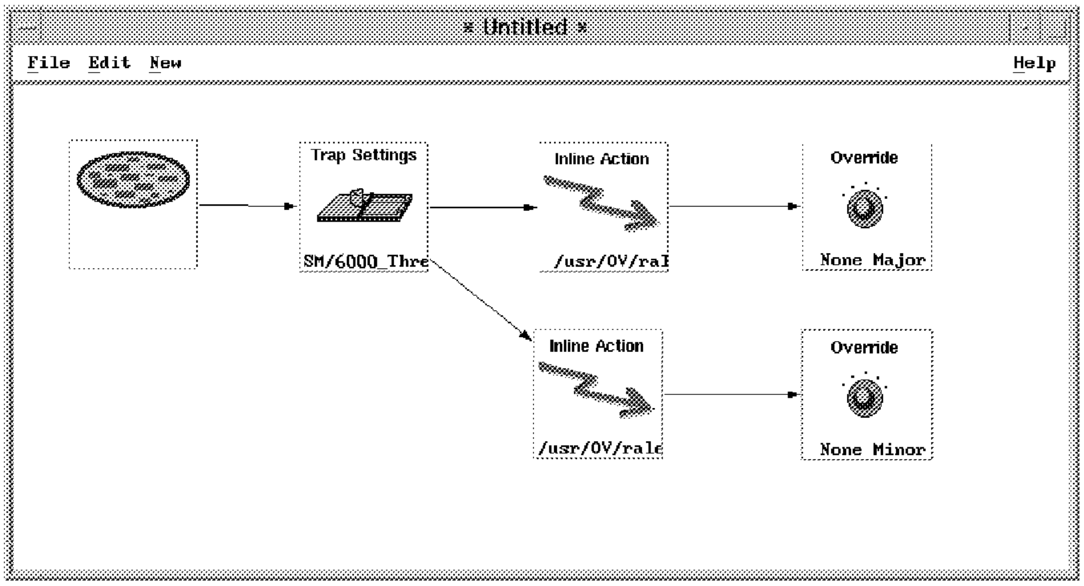
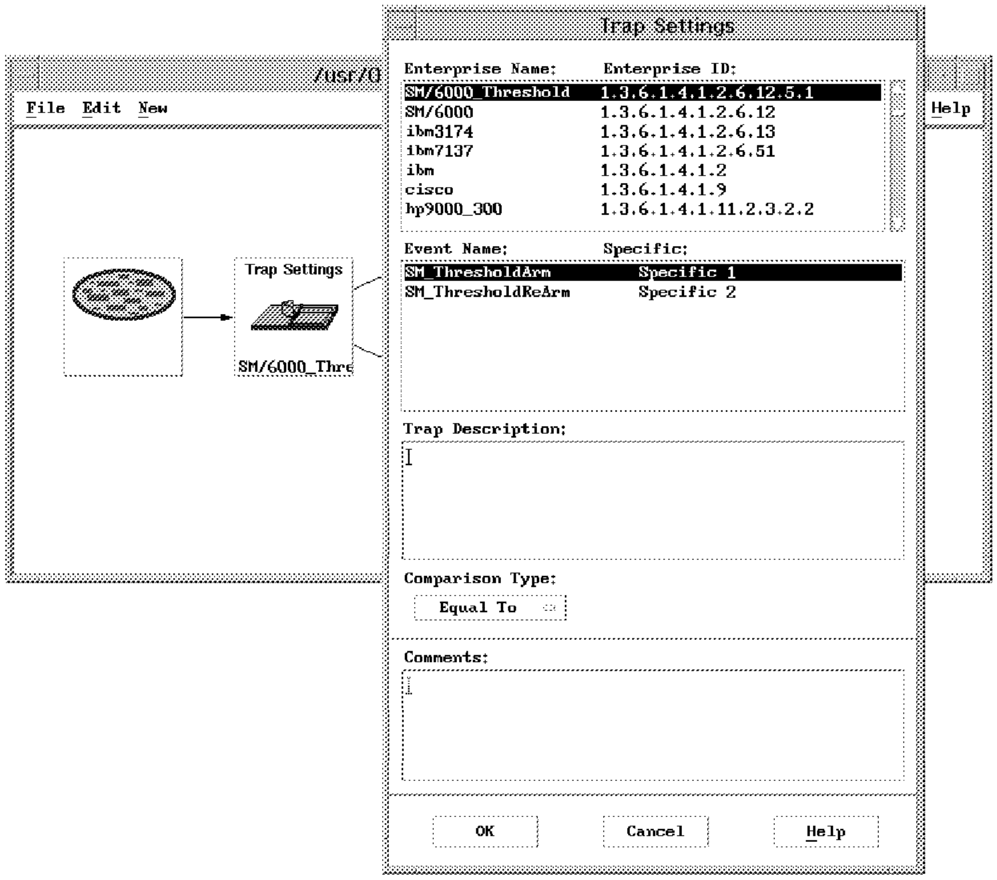


Figure 150. The Complete Ruleset

In order to construct this ruleset, we completed the following steps:

1. We set the default ruleset behavior to Pass.
2. Next we identified the System Monitor Threshold Arm Trap by using a Trap Setting node as shown in Figure 151.



Enterprise Name:	Enterprise ID:
SM/6000_Threshold	1.3.6.1.4.1.2.6.12.5.1
SM/6000	1.3.6.1.4.1.2.6.12
ibm3174	1.3.6.1.4.1.2.6.13
ibm7137	1.3.6.1.4.1.2.6.51
ibm	1.3.6.1.4.1.2
cisco	1.3.6.1.4.1.9
hp9000_300	1.3.6.1.4.1.11.2.3.2.2

Event Name:	Specific:
SM_ThresholdArm	Specific 1
SM_ThresholdReArm	Specific 2

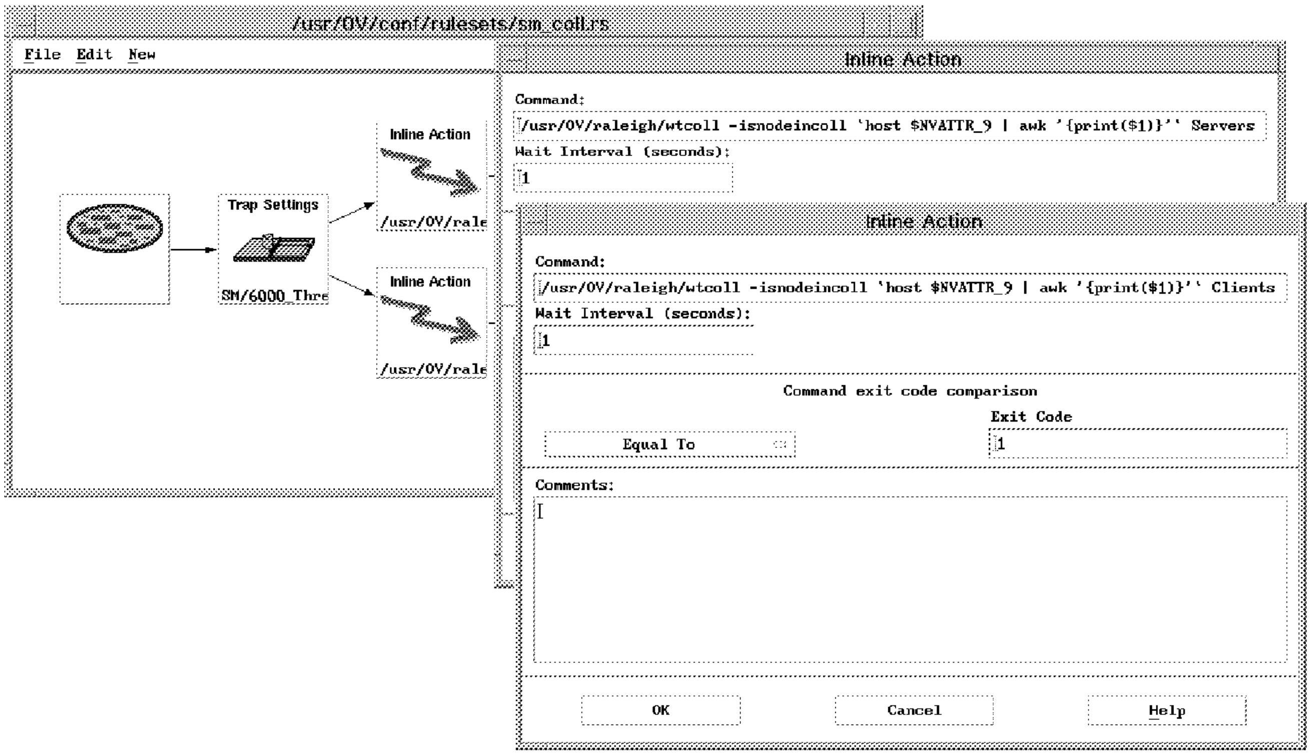
Trap Description:

Comparison Type: Equal To

Comments:

Figure 151. Detecting the System Monitor Threshold Arm Trap

3. Our next task was to enable the ruleset to identify if the trap came from a server node or a client node. To accomplish this, we added two Inline Action nodes both executing wtcoll. Figure 152 on page 180 shows how we configured these nodes.



Command:
```
/usr/OV/raleigh/wtcoll -isnodeincoll `host $NVATTR_9 | awk '{print($1)}'` Servers
```
Wait Interval (seconds): 1

Command:
```
/usr/OV/raleigh/wtcoll -isnodeincoll `host $NVATTR_9 | awk '{print($1)}'` Clients
```
Wait Interval (seconds): 1

Command exit code comparison

Equal To — Exit Code: 1

Comments:

Figure 152. Using Inline Action Nodes to Check for Collection Membership

The command format is rather complex:

```
wtcoll -isnodeincoll `host $NVATTR_9 | awk '{print($1)}'` Servers
```

The reason for this is that Systems Monitor always works with IP addresses, whereas the selection name in the NetView for AIX object database is the system name. Hence, we used the host command to convert from address to name.

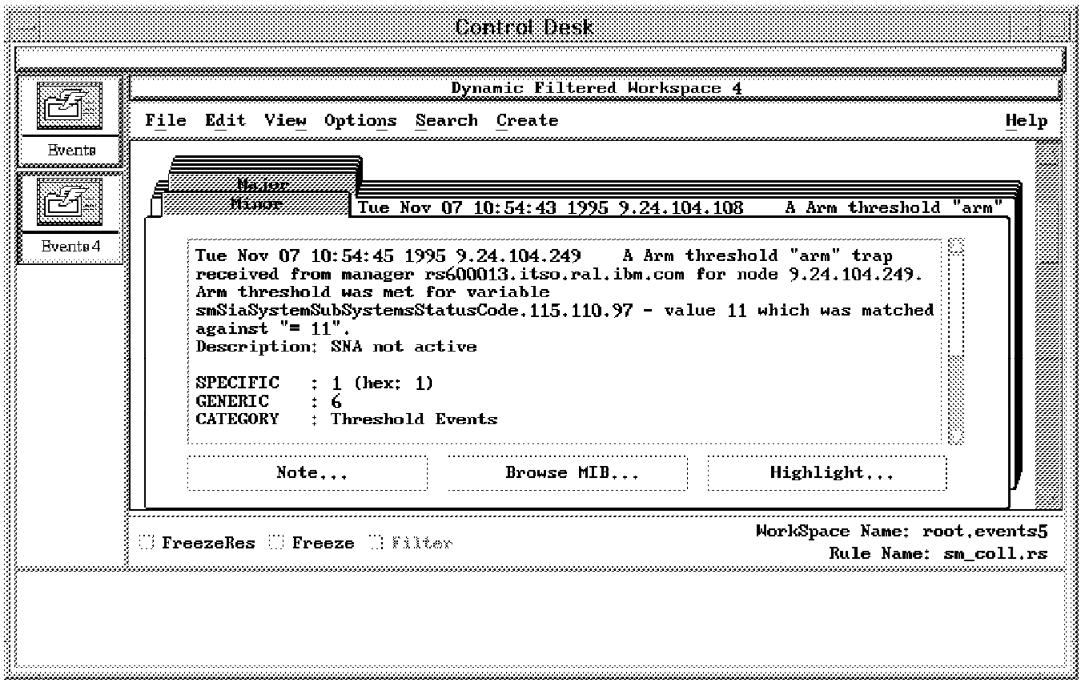
Note also that the threshold trap comes from the Systems Monitor for AIX Mid Level Manager (MLM), so the origin of the trap will be the MLM machine. To check the machine to which the threshold applies we had to specify the ninth variable in the trap.

4. Our final step was to add Override nodes to change the severity of the threshold event for the two collection types.

7.1.2.2 Testing the Manipulating Trap Severity Example

To test this example we created some Systems Monitor MLM threshold monitors using the Agent Policy Manager feature of NetView for AIX. The threshold was configured to be triggered when the SNA subsystem was found to be inactive on the target node. The process we used to do this is shown in 9.4, “An APM Example” on page 204. We elected to distribute the threshold to both the Servers and Clients collections.

The result of starting the threshold monitor is shown in Figure 153 on page 181. You can see that the trap for rsserver (9.24.104.108) has been changed to Major severity, and the trap for rs600019 (9.24.104.249) has been set to Minor.


Control Desk
Events
Events4
Dynamic Filtered Workspace 4
File Edit View Options Search Create Help
Major
Minor
Note... Browse MIB... Highlight...
FreezeRes Freeze Filter

```
Tue Nov 07 10:54:43 1995 9.24.104.108    A Arm threshold "arm"
```

```
Tue Nov 07 10:54:45 1995 9.24.104.249    A Arm threshold "arm" trap
received from manager rs600013.itso.ral.ibm.com for node 9.24.104.249.
Arm threshold was met for variable
smSiaSystemSubSystemsStatusCode.115.110.97 - value 11 which was matched
against "= 11".
Description: SNA not active

SPECIFIC   : 1 (hex: 1)
GENERIC    : 6
CATEGORY   : Threshold Events
```

```
WorkSpace Name: root.events5
     Rule Name: sm_coll.rs
```

Figure 153. Event Display With Override

7.1.3 Suppressing Subsidiary Events When a Router Is Down

A frequent problem in event management is the flood of events that result from the loss of one system such as a router. In these situations, the real interest is in the root cause and not the effects of the problem. Figure 154 on page 182 illustrates the problem.

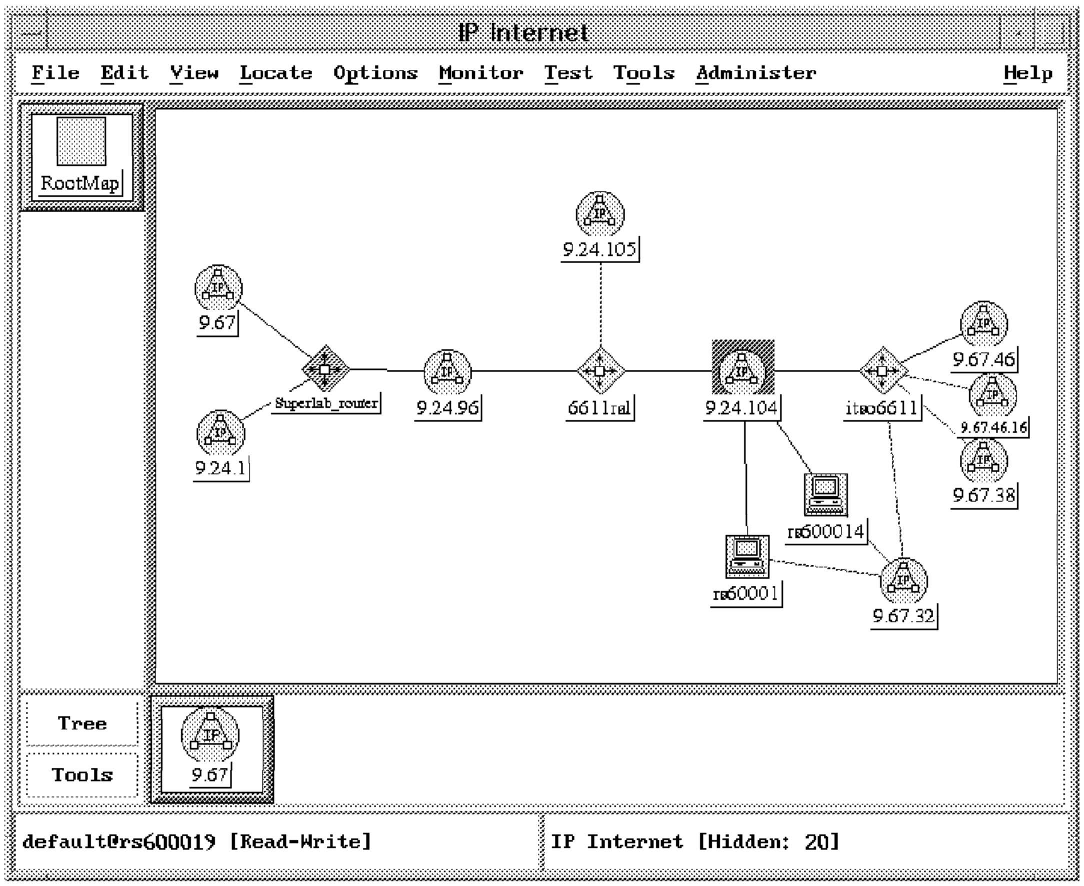


Figure 154. The Dependent Nodes Problem. In this network the NetView for AIX machine is in network 9.24.104. If the Superlab_router node were to fail there would be no route to the 9.67 and 9.24.1 networks, with the result that we would receive Node Down events for all the machines in those two networks. It would be better if those events could be suppressed while the router was down.

In this example, we used the Collection Facility in combination with event rulesets to demonstrate one method for dealing with this problem.

The method we used once again involved use of the wtcoll sample program. Using the Collection Facility, we defined a collection called nodes_dependent_on_router containing only those nodes that are in a particular router′s domain. Having done this we need a ruleset that will do the following two things for us:

1. Provide a trigger mechanism so that message suppression is activated when the router fails. We chose to use a ruleset global variable to signal this.
2. Suppress Node Down and Interface Down messages from nodes in the dependent subnets once the trigger has fired.

7.1.3.1 Creating the Ruleset

The complete ruleset that we constructed for this example is shown in Figure 155 on page 183.

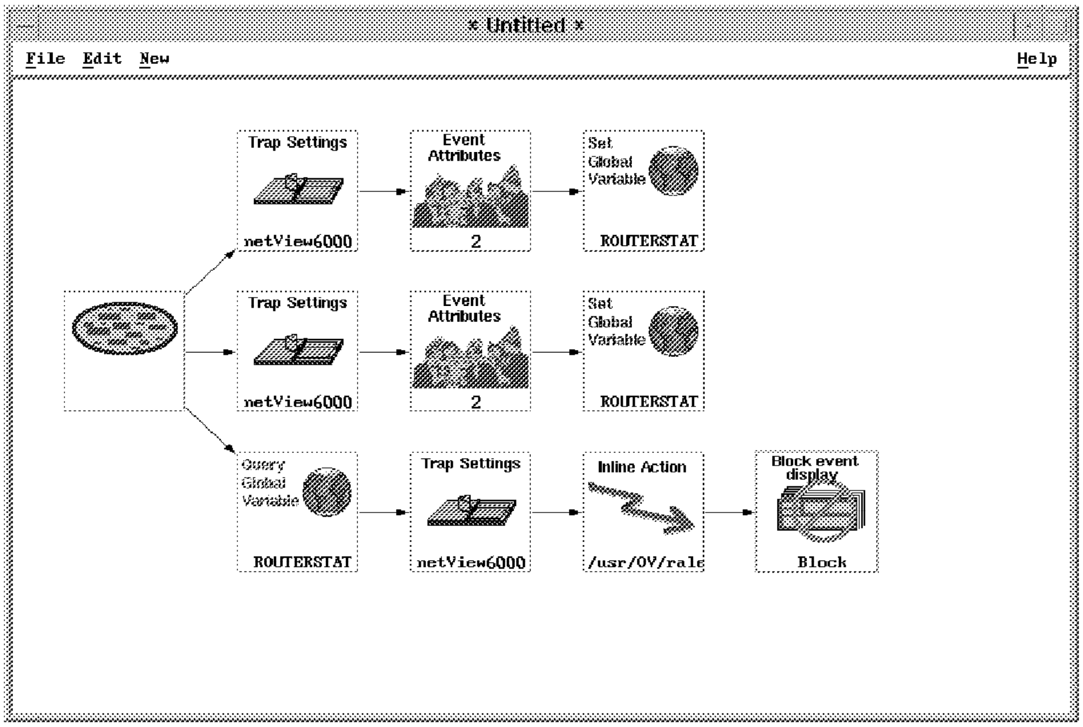
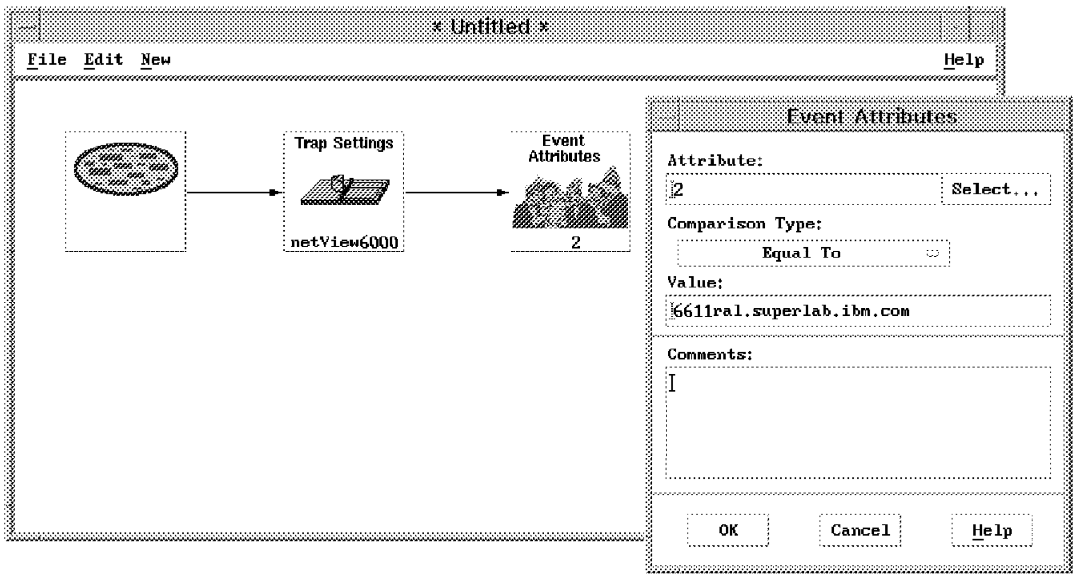


Figure 155. The Complete Dependent Node Suppression Ruleset

To construct this ruleset, we completed the following steps:

1. The first step is to set the default behavior of the ruleset to Pass (see Figure 59 on page 89).
2. Next we added a Trap Settings node to check for Node Down events (see 6.2.1, "Clearing Outstanding Events via Correlation" on page 92 for an example of this).
3. To the Trap Settings node we added an Event Attributes node, to look for events coming from the router. Figure 156 on page 184 shows the configuration of this.



Attribute: 2

Comparison Type: Equal To

Value: 6611ral.superlab.ibm.com

Comments:

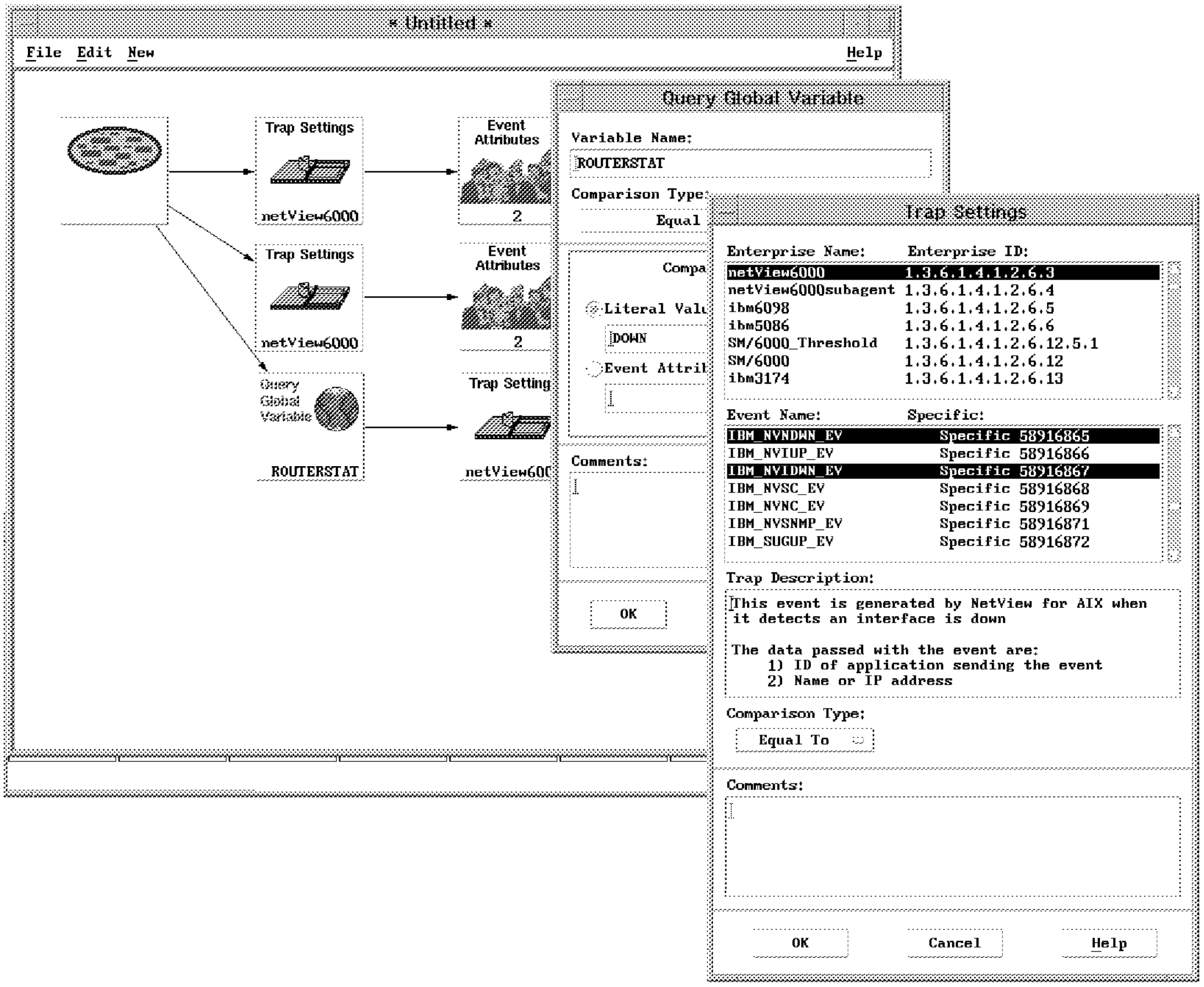
Figure 156. Adding the Event Attributes Node

4. Completing this strand is a Set Global Variable node, which sets the value of $ROUTERSTAT to *DOWN*.

5. We repeated the sequence of nodes in steps 2 to 4 but this time checked for Node Up in Trap Settings and changed the value of the global variable to UP. This completes the trigger side of the ruleset.

6. We only want the remainder of the ruleset to be active when the $ROUTERSTAT global variable is set to DOWN. The first node in the final strand is therefore a Query Global Variable.

7. Next we added a Trap Settings node that checks for *either* a Node Down or an Interface Down event. Figure 157 on page 185 shows the configuration of this node and also of the Query Global Variable node.



Variable Name: ROUTERSTAT
Comparison Type: Equal
Literal Value: DOWN
Event Attribute:
Comments:

Enterprise Name:	Enterprise ID:
netView6000	1.3.6.1.4.1.2.6.3
netView6000subagent	1.3.6.1.4.1.2.6.4
ibm6098	1.3.6.1.4.1.2.6.5
ibm5086	1.3.6.1.4.1.2.6.6
SM/6000_Threshold	1.3.6.1.4.1.2.6.12.5.1
SM/6000	1.3.6.1.4.1.2.6.12
ibm3174	1.3.6.1.4.1.2.6.13

Event Name:	Specific:
IBM_NVNDWN_EV	Specific 58916865
IBM_NVIUP_EV	Specific 58916866
IBM_NVIDWN_EV	Specific 58916867
IBM_NVSC_EV	Specific 58916868
IBM_NVNC_EV	Specific 58916869
IBM_NVSNMP_EV	Specific 58916871
IBM_SUGUP_EV	Specific 58916872

Trap Description:

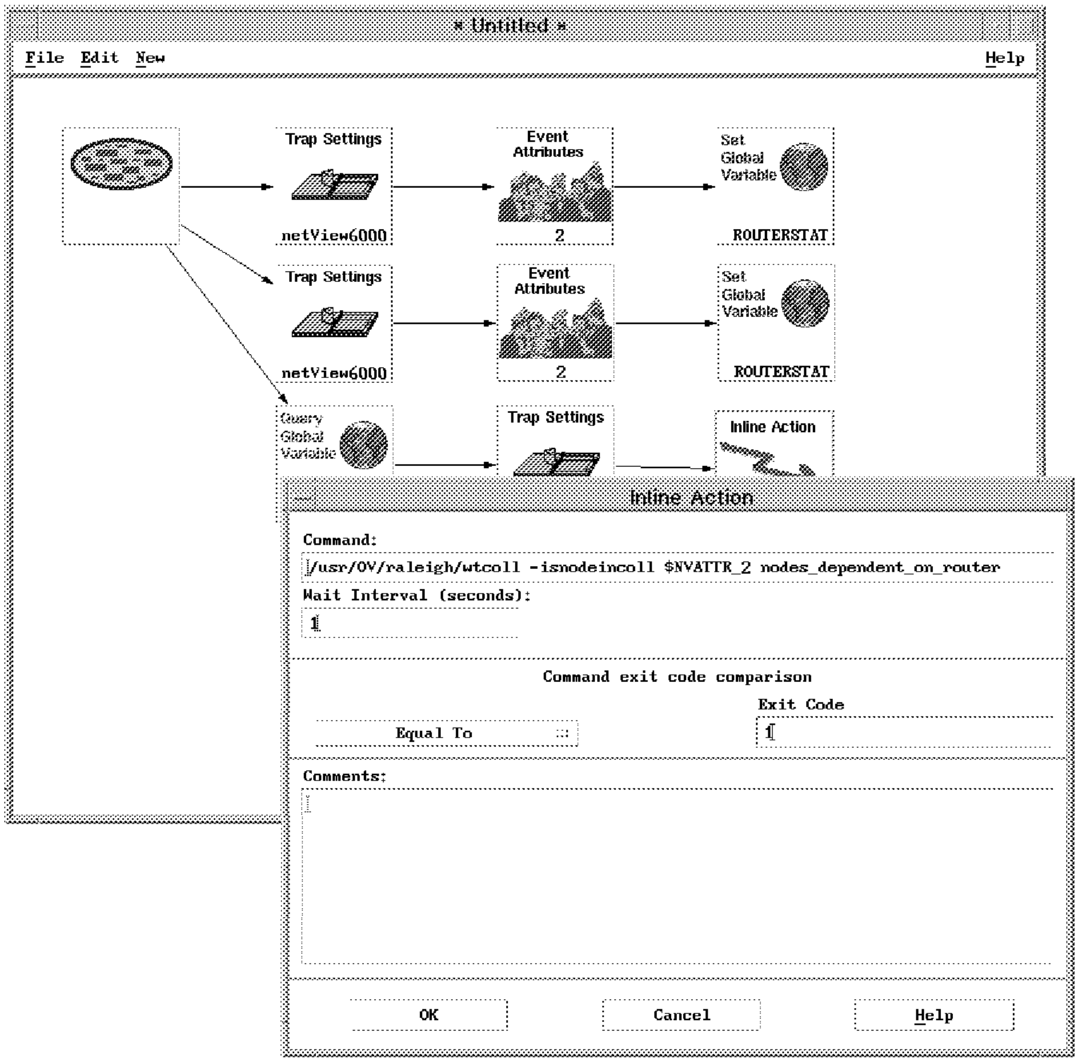
```
This event is generated by NetView for AIX when
it detects an interface is down

The data passed with the event are:
     1) ID of application sending the event
     2) Name or IP address
```

Comparison Type: Equal To

Comments:

Figure 157. Adding the Trap Settings Nodes

8. Next we wanted to check to see if the node or interface down event was reported for a node with connectivity dependent on the failed router. As in the previous example we used an Inline Action node to execute the wtcoll sample program. Figure 158 on page 186 shows you the configuration of this node.



Command:

```
/usr/OV/raleigh/wtcoll -isnodeincoll $NVATTR_2 nodes_dependent_on_router
```

Wait Interval (seconds): 1

Command exit code comparison

Equal To

Exit Code: 1

Comments:

Figure 158. Adding the Inline Action Node

9. Finally we added the Block Event Display node to prevent display of the event card for the dependent node.

7.1.3.2 Testing the Router Down Example

In order to test our example, we completed the following steps:

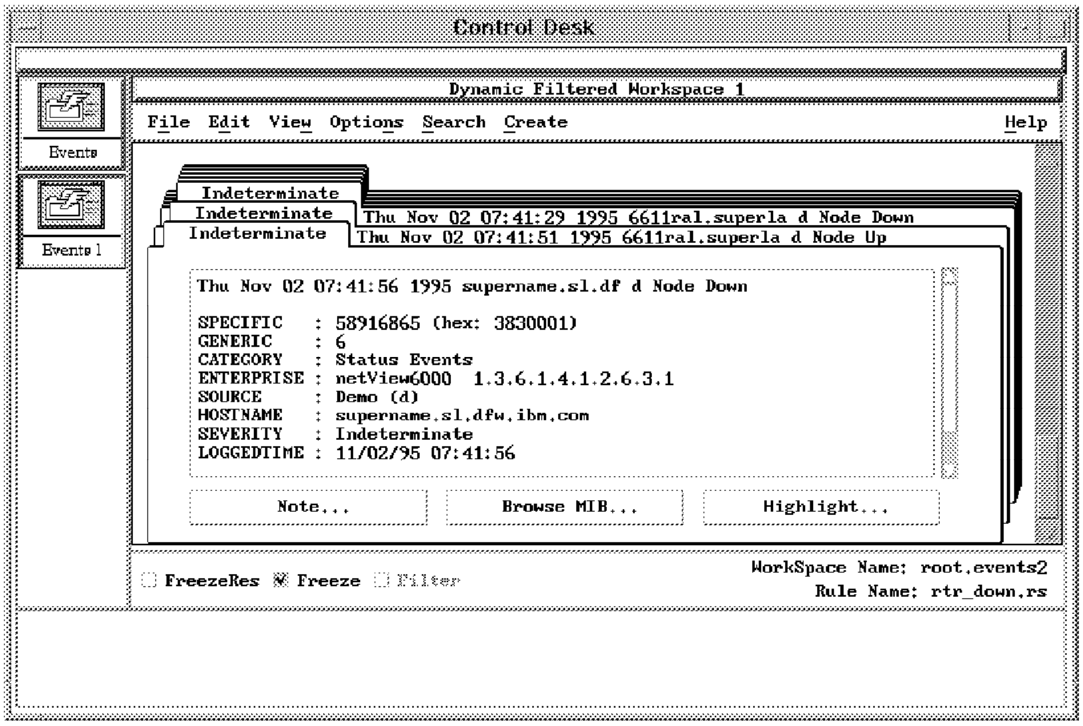
1. We opened a new dynamic workspace using the rule_coll.rs file.
2. We sent a Node Down event for our defined router using the command:

   ```
   event -h 6611ral.superlab.ibm.com -e NDWN_EV
   ```

3. Then we sent a Node Down event for a node in the 9.24.1 network using the following command:

   ```
   event -h supername.sl.dfw.ibm.com -e NDWN_EV
   ```

   And nothing appeared in our workspace.
4. We then reversed the process by sending a Node Up event for the router.
5. Again we sent the node down event for node supername and this time, as expected, the event appeared. The resulting workspace events are shown in Figure 159 on page 187.


Control Desk
Dynamic Filtered Workspace 1
File Edit View Options Search Create Help
Events
Events 1
Indeterminate
Indeterminate
Indeterminate
Note... Browse MIB... Highlight...
FreezeRes Freeze Filter

```
Thu Nov 02 07:41:29 1995 6611ral.superla d Node Down
Thu Nov 02 07:41:51 1995 6611ral.superla d Node Up

Thu Nov 02 07:41:56 1995 supername.sl.df d Node Down

SPECIFIC   : 58916865 (hex: 3830001)
GENERIC    : 6
CATEGORY   : Status Events
ENTERPRISE : netView6000  1.3.6.1.4.1.2.6.3.1
SOURCE     : Demo (d)
HOSTNAME   : supername.sl.dfw.ibm.com
SEVERITY   : Indeterminate
LOGGEDTIME : 11/02/95 07:41:56

WorkSpace Name: root.events2
Rule Name: rtr_down.rs
```

Figure 159. Detecting a Node Down Event

7.1.4 The wtdepend Sample Application

The rule_coll.rs ruleset described in the last example (7.1.3, “Suppressing Subsidiary Events When a Router Is Down” on page 181) shows promise as a technique for reducing event clutter. However, for practical purposes it falls short in the following areas:

- It is unique to a specific router. A generic solution would be better.
- It can only handle one router failure at a time.
- It is triggered by a complete router failure, whereas really access to other nodes is dependent on the availability of individual router *interfaces*.
- Configuring it is a very manual process.
- Even when the events are suppressed, the node status still changes to Critical (the symbol goes red). A more accurate status would be Unknown.

The final shortcoming of rule_coll.rs is less obvious. It does a good job of blocking node and interface down events from dependent nodes. However, those events do not *really* come from the subject nodes at all. They are generated by the NetView for AIX netmon daemon as a result of poll failures. It is a general rule of automation and event handling that you should take action as near to the source as possible. Therefore, instead of polling a node that we know is unreachable (which involves timeouts and retries which may interfere with regular polling), it would be better to suspend polling for the affected nodes until the router has recovered.

We decided to produce an application to implement the dependent node failure in a way that overcomes these problems. The result of this is the wtdepend package.

7.1.4.1 Elements of the wtdepend Package

There are three components to wtdepend, as follows:

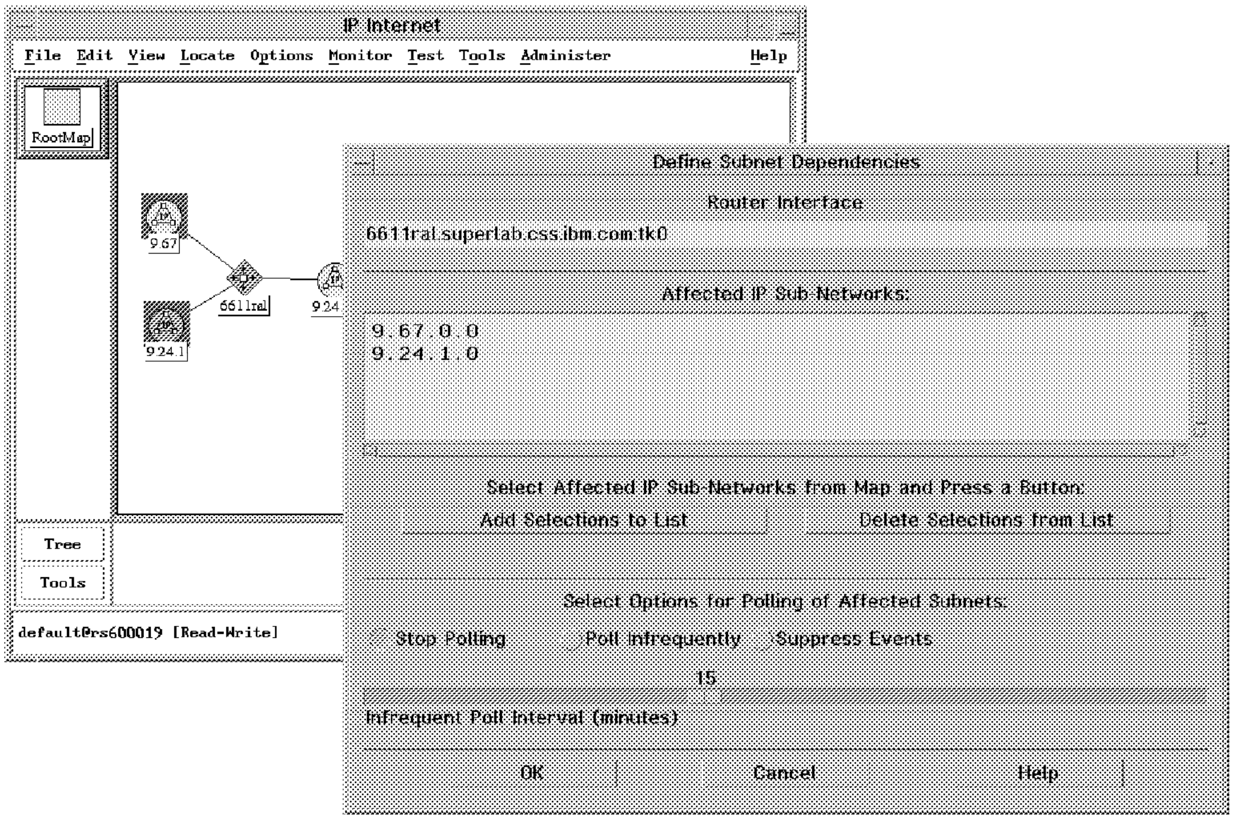
- An EUI dialog that simplifies the definition of the collection of nodes that is dependent on a given router interface.
- A trigger ruleset (wtdepend.rs) which runs in the background. It signals the failure of a router interface and optionally alters netmon polling parameters for the dependent nodes.
- A display ruleset (wtdepdisp.rs) which is activated when the trigger has fired, and which overrides node status and event severity for the dependent nodes.

If you want to get a copy of the package, refer to the instructions in Appendix A, “How to Obtain the Samples in this Book” on page 257.

7.1.4.2 Operation of the wtdepend Package

EUI Dialog: Having installed the package by following the instructions in the README file, the first thing to do is to define the dependencies, as follows:

1. Navigate to the IP Internet submap.
2. Select a router interface (that is, the line connecting a router to a subnet, not the router symbol itself).
3. Select Tools→Define Router Interface Dependencies from the menu bar. A dialog screen will appear.
4. Select the subnets that will be unreachable if the router interface fails. Then click on **Add Selections to List**. Figure 160 shows an example of this.


IP Internet
File Edit View Locate Options Monitor Test Tools Administer Help
RootMap
9.67
6611ral
9.24
9.24.1
Tree
Tools
default@rs600019 [Read-Write]
Define Subnet Dependencies
Router Interface
Affected IP Sub Networks:
Select Affected IP Sub Networks from Map and Press a Button:
Add Selections to List
Delete Selections from List
Select Options for Polling of Affected Subnets:
Stop Polling
Poll Infrequently
Suppress Events
Infrequent Poll Interval (minutes)
OK
Cancel
Help

6611ral.superlab.css.ibm.com:tk0

```
9.67.0.0
9.24.1.0
```

15

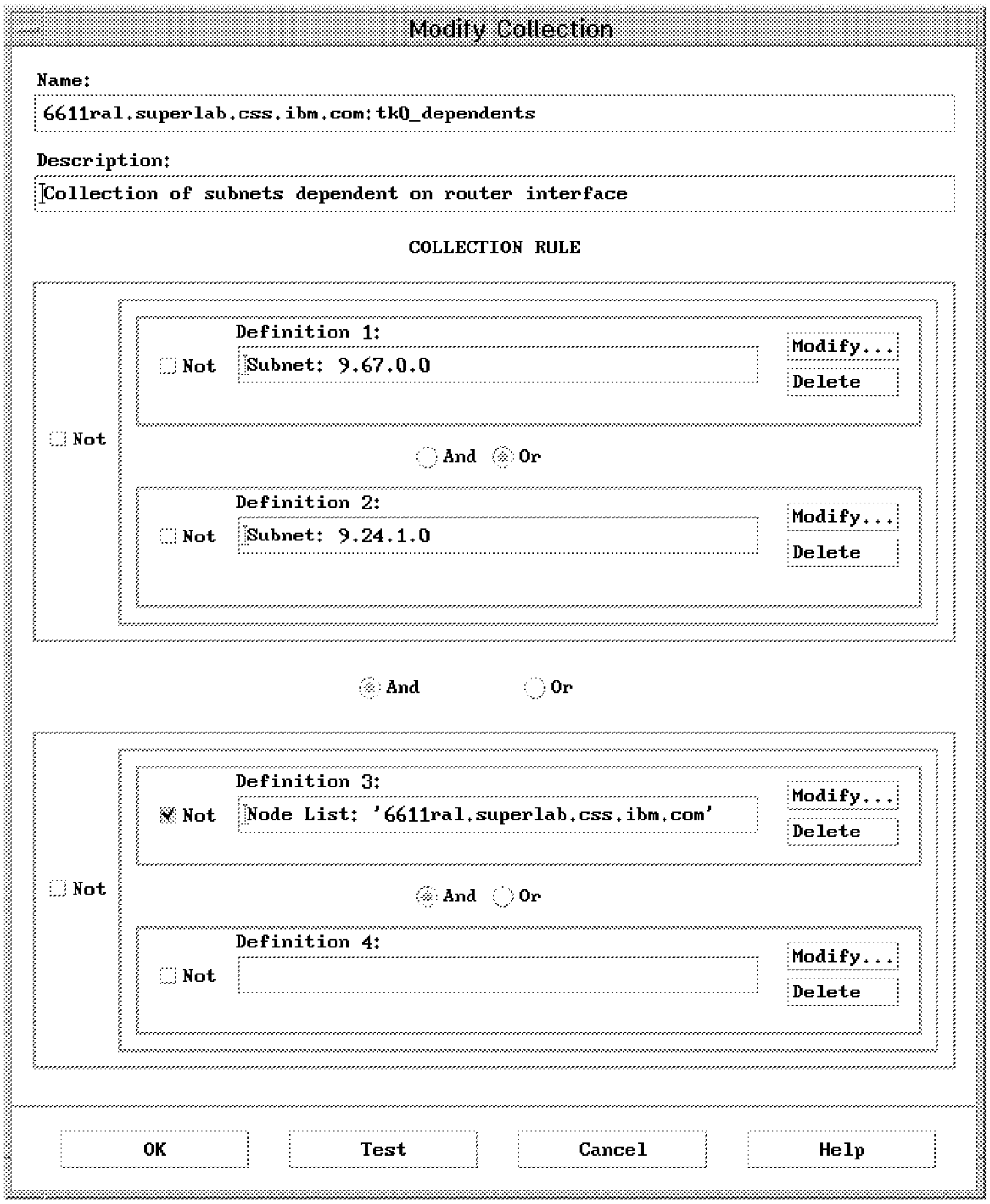
Figure 160. wtdepend: Defining Interface Dependencies

5. Define the policy that you want to follow for the dependent nodes (to poll less frequently, or never, or simply to suppress Node Down and Interface Down events).

6. Click on **OK**.

At this point the program does the following three things:

1. It creates a collection containing the dependent subnets that you indicated in the dialog. Figure 161 shows the collection created by the dialog shown in Figure 161.



Modify Collection

Name:
6611ral.superlab.css.ibm.com:tk0_dependents

Description:
Collection of subnets dependent on router interface

COLLECTION RULE

Not

Definition 1: Not — Subnet: 9.67.0.0

And [✓ Or]

Definition 2: Not — Subnet: 9.24.1.0

[✓ And] Or

Not

Definition 3: [✓ Not] — Node List: '6611ral.superlab.css.ibm.com'

[✓ And] Or

Definition 4: Not

Figure 161. wtdepend: A Dependent Subnet Collection

2. It adds a field named dependent_collection_exists to the object database record for the router, and sets the value to TRUE.

3. It creates a further object database field called dependent_coll_command in the record of the interface that you selected.

wtdepend.rs Ruleset: The wtdepend.rs ruleset is automatically added to /usr/OV/conf/ESE.automation by the install process. The complete ruleset is shown in Figure 162.

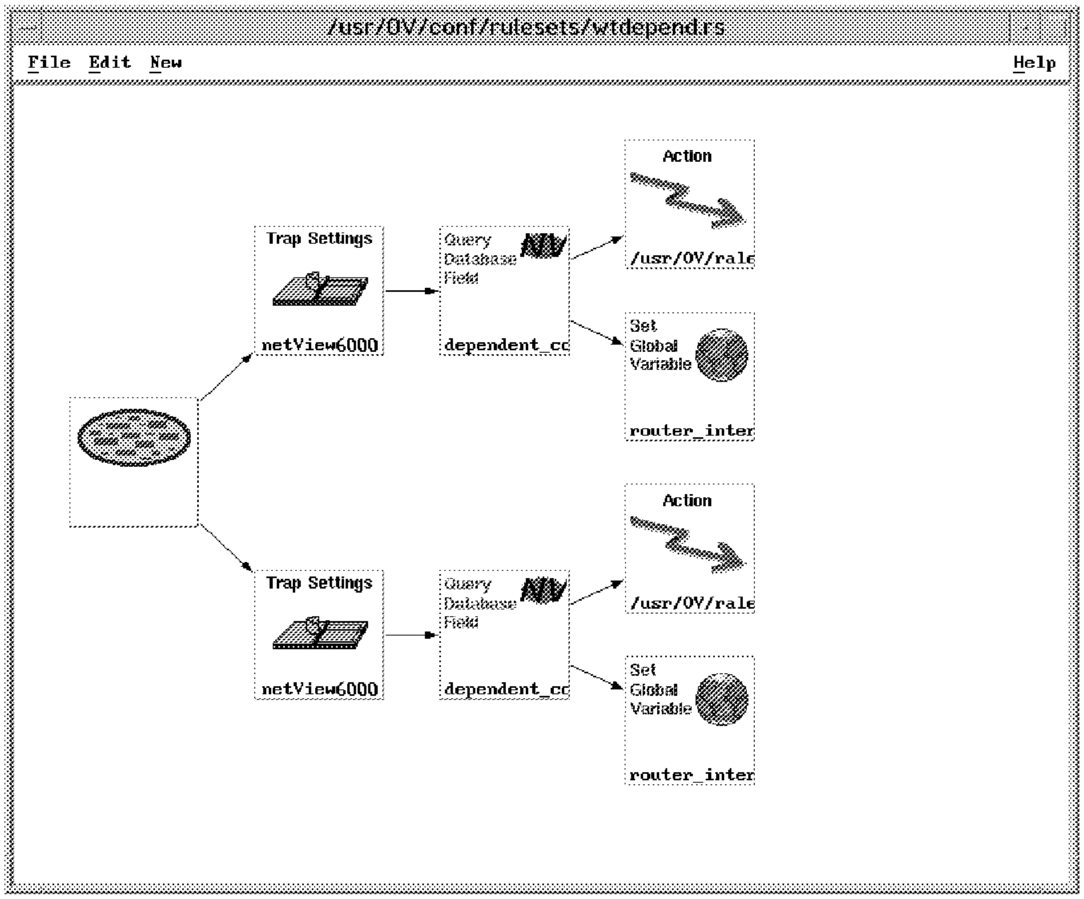


Figure 162. The wtdepend.rs Ruleset

The operation of this ruleset is as follows:

1. First a Trap Settings node checks for Interface Down events.
2. Next it uses a Query Database node to see if the dependent_collection_exists field for the router is set to TRUE. If so, the two final nodes come into play.
3. The Set Global Variable node increments a numeric variable, router_interface_failures. This acts as a counter of the number of router interfaces with dependent nodes that are currently down.
4. The Action node executes a shell script. This extracts the name of the router interface from the event message and then reads the value of the dependent_coll_command field. It then executes this value as a command. At present the command does one or two things, as follows:
    a. If you specified that the netmon polling interval for the dependent nodes should be increased, it will use wtcoll to issue xnmsnmpconf commands that achieve this.

   b. It always executes a program called wtdepend_list. This program maintains a collection called unreachable_nodes. This is a *collection of collections*. That is, it combines the dependent node collections of all router interfaces that are currently down. The source code for wtdepend_list is in Appendix C, “C Code for the wtdepend_list Sample Program” on page 267.

5. The second strand of the wtdepend.rs ruleset reverses the process. That is to say: if it detects an Interface Up event from a router with dependent node collections defined, it will decrement the router_interface_failures counter, remove the dependent collection from the unreachable_nodes collection and reset the polling intervals.

wtdepdisp.rs Ruleset: The wtdepdisp.rs ruleset is designed to be invoked in a dynamic workspace. The complete ruleset is shown in Figure 163 on page 192.

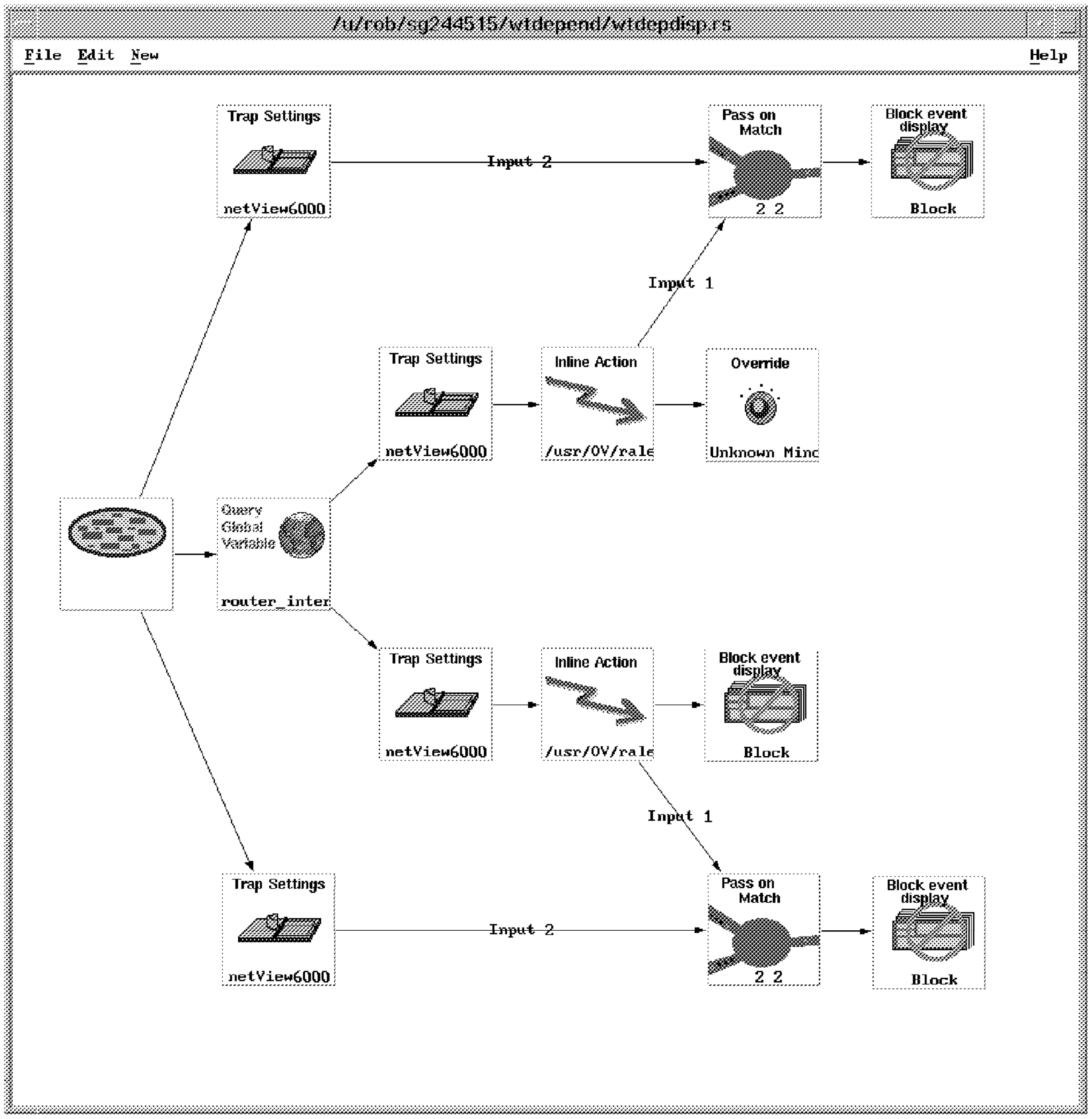


Figure 163. The wtdepdisp.rs Ruleset

The operation of this ruleset is as follows:

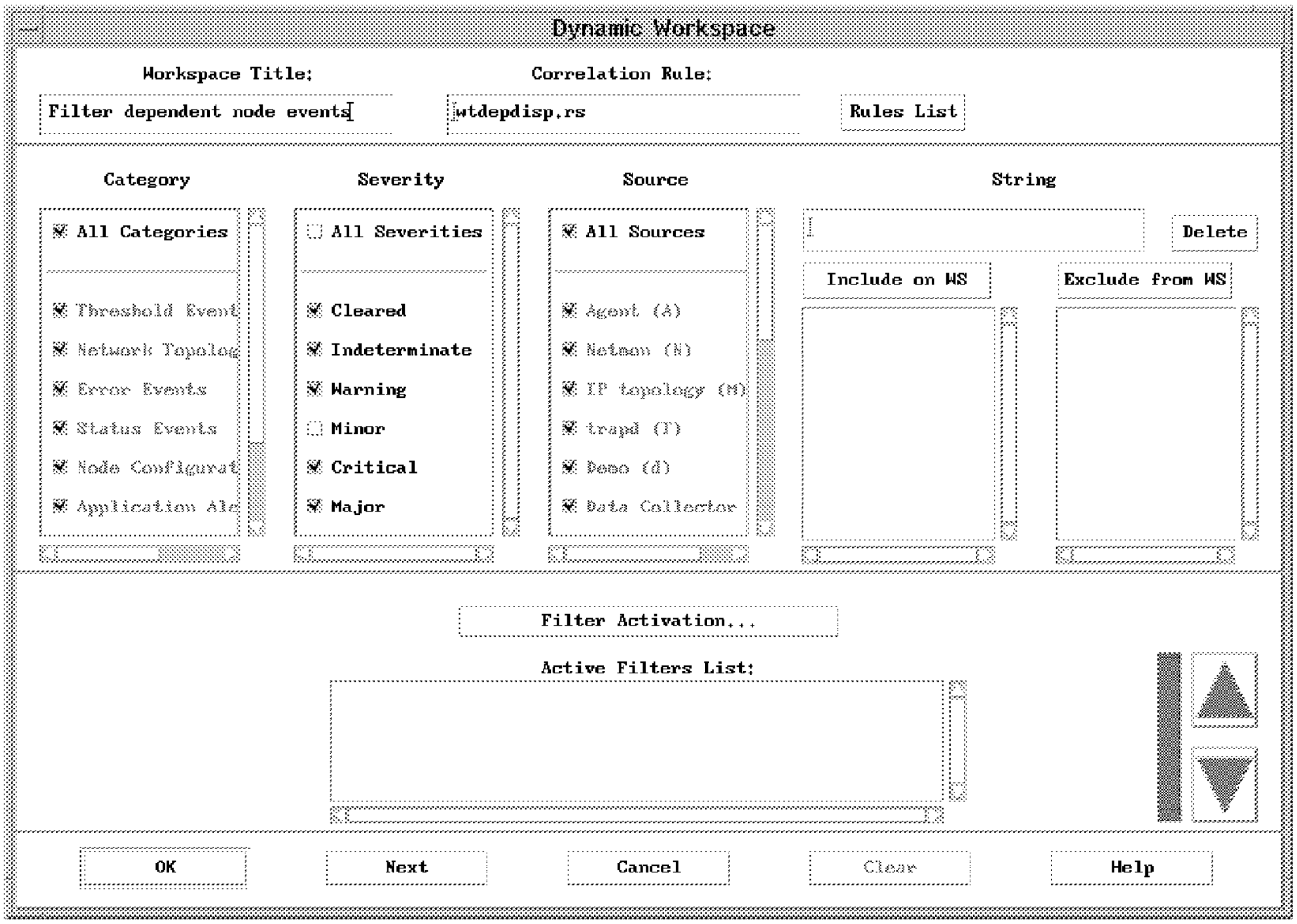
1. The ruleset contains four paths, for handling Node and Interface Down and Up events.
2. In the Down path it first uses a Query Global Variable node to check the value of router_interface_failures to see if it is greater than zero. This makes the ruleset very efficient, since the rest of the nodes are ignored in normal circumstances.
3. Next it checks for event type, using two Trap Settings nodes. These are configured to pass Node Down and Interface Down events.

4. The next nodes are Inline Actions that uses wtcoll to check if the subject node is in collection unreachable_nodes. This is the collection of dependent node collections that was updated by the wtdepend.rs ruleset. Any node in this collection is expected to be unreachable, because a router in the path to it has failed.

5. If the node is in the unreachable_nodes collection two things happen:

   a. There is a node which blocks the event. For Interface Down this is a simple Block Event Display node, but for Node Down an Override node changes the node status to Unknown (because we don′t know if it is really down or not) and sets the event severity to Minor. Ideally we would want to block the event completely, but the node status override only works if the event is passed to nvevents, which means that we cannot use a Block Event Display node. The circumvention to this is to limit the event severities that we choose to display when we create the dynamic workspace. Figure 164 shows the creation of this workspace, in which we both select the ruleset to invoke and choose not to display Minor events.

   b. A Pass on Match node is initialized which will wait up to one hour for the corresponding Node Up and Interface Up.



Workspace Title: Filter dependent node events

Correlation Rule: wtdepdisp.rs

Category	Severity	Source
☒ All Categories	☐ All Severities	☒ All Sources
☒ Threshold Event	☒ Cleared	☒ Agent (A)
☒ Network Topolog	☒ Indeterminate	☒ Netmon (N)
☒ Error Events	☒ Warning	☒ IP topology (M)
☒ Status Events	☐ Minor	☒ trapd (T)
☒ Node Configurat	☒ Critical	☒ Demo (d)
☒ Application Ale	☒ Major	☒ Data Collector

String

Active Filters List:

Figure 164. Creating the Dynamic Workspace for the wtdepend Sample

6. The Node Up and Interface Up paths of wtdepdisp.rs provide the resolving input for the Pass on Match nodes. You may want to refer to 6.2.1, “Clearing Outstanding Events via Correlation” on page 92 for a description of how this

operates. We chose this processing because the Up events are expected to occur after the failed router interface has returned. You can think of the Pass on Match as a way of remembering that the node *was previously* in the unreachable_nodes collection.

7.1.4.3 Testing wtdepend

It is rather disruptive to disable a router interface, so we chose to create an unreachable network by setting an invalid IP route for the network. Note, however, that in a dynamic routing environment this may not work. The steps to testing wtdepend were:

1. Create a dependent node collection for a router interface, as described in "EUI Dialog" on page 188.
2. Set up a dynamic workspace with the wtdepdisp.rs ruleset invoked, as described in "wtdepdisp.rs Ruleset" on page 191.
3. Change the polling frequency of the affected router to a short period (to improve the chance of a router interface failure being detected before the dependent failures).
4. Make the affected subnet unreachable, using the TCP/IP `route` command.

Chapter 8. Implementation Recommendations for Rulesets

In Chapter 6, “Examples of NetView for AIX Rulesets” on page 85 and Chapter 7, “Using the Collection Facility with Event Rulesets” on page 171, we have shown many examples of ways in which you can use event rulesets. The objective of all this function is to introduce some intelligence and automation into the network monitoring process.

Automation, however, means different things to different people. If you are to successfully attack the event flow in NetView for AIX you need to have a clear definition of what the objectives are and the path by which you can achieve them. In this section we will offer our suggestions of what the objectives should be, the path to follow and some approaches you can take to start down that path.

8.1 Objectives of Event Processing

Any event that we are interested in should be capable of being placed into one of the following three categories:

Error Event Something that provides information about a problem

Resolving Event Something that provides information about the resolution of a problem

Informational Event Something that provides supporting information, that may alter the impact or meaning of an event in the other two categories

If we view these categories as the fodder for our event processing system, there are several things we want to do with them, as follows:

- Ensure that they are only presented to users to whom they have a meaning
- Arrange for important events to be delivered by appropriate alert mechanisms (for example, pager or email)
- Arrange for resolving events to be paired with the problem events they resolve
- Implement *reactive automation* of events for which a particular response is always appropriate
- Implement *intelligent automation* of events for which the response requires some knowledge of policy or environment

Event rulesets give us a powerful set of tools with which to attack each of these objectives (the last objective really needs additional programming to tackle effectively). However, before we start on this path we should first look at the event stream we are dealing with.

8.2 Tackling the Event Stream

We have divided the events we want to handle into three categories, Error, Resolving and Informational. However, there is a fourth category which makes up a large proportion of the events in a typical NetView for AIX installation, that is, *unwanted* events. Some of these events are quite easy to recognize. For example, node and interface up and down events for user workstation devices often fall into this category. Others are less easy to predict, depending on the components in the network. If we are to do an effective job of handling events, filtering out the unwanted ones has to be the first priority, partly so that we can present the user with meaningful data and partly because we don′t want the event processing functions to be overwhelmed. With this in mind, we can map out the following path to follow in our quest for good event handling:

1. Suppress. The first thing to do is eliminate the unwanted events.
2. Route. Send the remaining events only to the people who are interested in receviving them.
3. Throttle. Eliminate repetitious event streams.
4. Correlate. Match failure with recovery to simplify the picture.
5. Automate. Handle events automatically.

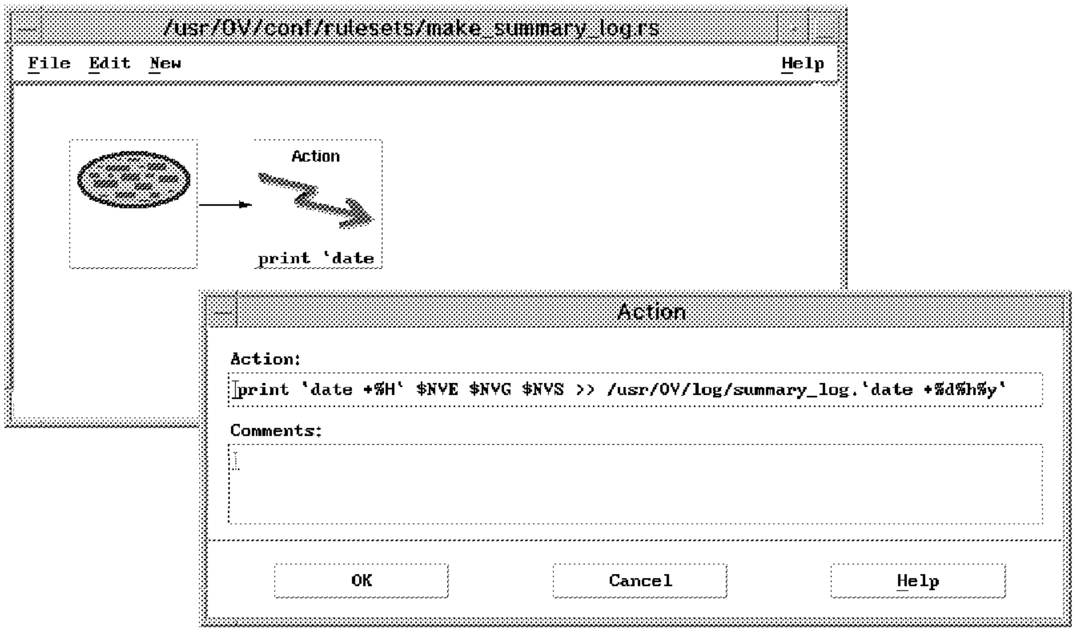
In all of this we need to remember the maxim that underlies all automation processing: *take action as near to the source of the event as possible*. This means that ruleset processing (at least in its present form) is usually the last choice method to attack a problem, since it occurs relatively late in the processing path. It is better to remove unwanted events by configuring them in trapd.conf, rather than using a ruleset, if the facilities in trapd.conf are capable of the task. Even better is to avoid generating the event in the first place. The Interface Down event is a good example of this. This apparently simple event represents a lot of processing and waiting by the netmon daemon. It is better to avoid generating the Interface Down for unimportant nodes, by choosing not to poll the nodes, than to generate it and then filter it out. You can cease polling an IP device by unmanaging it in all the maps.

We can learn a lot about event handling from experiences in MVS operations automation. Before MVS operators started to use consolidated console displays (some five to ten years ago), a typical control area would have numerous console screens, with continuous streams of messages coming from them. Only a small proportion of the messages could be read by the operator and acted on. With the advent of automation products such as AOC/MVS, MVS operations shops started to suppress, route, correlate and automate. The results of this were often spectacular, with 90% plus reductions in the quantity of messages received by the operators.

The first step in this effort was to analyze the event flow, to identify the most frequently-occurring messages. We recommend that you take a similar approach when configuring NetView for AIX event processing.

8.2.1 Know the Enemy

One advantage we have in the world of NetView for AIX compared with the MVS console messages is that all the event types are uniquely identified by their SNMP elements: Enterprise ID, Generic ID and Specific ID. Unfortunately, the standard event log, trapd.log, fails to record these key elements, thus making analysis more difficult. We chose to use an automation ruleset to write the data out to a log file in a format that is easier to handle. Figure 165 on page 197 shows this ruleset.



Action:

```
print ‘date +%H‘ $NVE $NVG $NVS >> /usr/OV/log/summary_log.‘date +%d%h%y‘
```

Comments:

Figure 165. The make_summary_log Ruleset

Having collected the event informatin into a log, we wrote a relatively simple shell script to analyze it. Figure 166 shows the shell script.

```
USAGE=″usage: $0 ·mmdd“″  #Script to sort a logfile by activities

if (($# > 1))
then
  print ″You passed too many arguments to $0.″
  print ″$USAGE″
  exit 1
elif (($# == 1))
then
  logfile=$1
else
  read logfile?′Enter the date to examine (form = mmdd) ==> ′
fi


if ·· ! -f /usr/OV/log/summary_log.$logfile ““
then
  print ″     File /usr/OV/log/summary_log.$logfile does not exist!″
  print ″Exiting...″
  exit 1
fi

integer hour
integer total_traps
integer count
integer prev_hour
```

Figure 166 (Part 1 of 2). logsort Shell Script for Analyzing the Ruleset-Created Trap Log

```
# First sort the file, remove duplicates and add duplicate counts

sort /usr/OV/log/summary_log.$logfile | uniq -c > /tmp/summary_log.sorted

# Read the file and print hourly summaries

exec 3< /tmp/summary_log.sorted
read -u3 total_traps hour mibID generic_trap specific_trap
((prev_hour = hour))
print "=== Event Statistics for hour $hour ============================="
print
printf "%-6s%-30s%-11s%s\n" Count Enterprise_ID Generic_ID Specific_ID
printf "%-6s%-30s%-11s%s\n" $total_traps $mibID $generic_trap $specific_trap
while read -u3 count hour mibID generic_trap specific_trap
do
  if ((hour == prev_hour))
  then
    printf "%-6s%-30s%-11s%s\n" $count $mibID $generic_trap $specific_trap
    ((total_traps = total_traps + count))
  else
    print "Total Traps: $total_traps"
    print
    print "=== Event Statistics for hour $hour ============================="
    print
    printf "%-6s%-30s%-11s%s\n" Count Enterprise_ID Generic_ID Specific_ID
    printf "%-6s%-30s%-11s%s\n" $count $mibID $generic_trap $specific_trap
    total_traps=count
    ((prev_hour = hour))
  fi
done
print "Total Traps:     $total_traps"

rm /tmp/summary_log.sorted
```

Figure 166 (Part 2 of 2). logsort Shell Script for Analyzing the Ruleset-Created Trap Log

Figure 167 on page 199 shows a sample of the output produced by this logsort shell script.

```
=== Event Statistics for hour 12 =============================

Count Enterprise_ID                 Generic_ID Specific_ID
1     1.3.6.1.4.1.2.6.3.1           6          59179056
1     1.3.6.1.4.1.2.6.3.1           6          59179057
4     1.3.6.1.4.1.2.6.3.1           6          1357
Total Traps: 6

=== Event Statistics for hour 13 =============================

Count Enterprise_ID                 Generic_ID Specific_ID
4     1.3.6.1.4.1.2.6.3.1           6          1358
3     1.3.6.1.4.1.2.6.3.1           6          58916865
3     1.3.6.1.4.1.2.6.3.1           6          58916867
3     1.3.6.1.4.1.2.3.1.2.1.1.3     4          0
Total Traps: 13

=== Event Statistics for hour 14 =============================

Count Enterprise_ID                 Generic_ID Specific_ID
1     1.3.6.1.4.1.2.6.3.1           6          1357
1     1.3.6.1.4.1.2.6.3.1           6          1358
1     1.3.6.1.4.1.2.6.3.1           6          50790445
1     1.3.6.1.4.1.2.6.3.1           6          58785793
1     1.3.6.1.4.1.2.6.3.1           6          58785795
2     1.3.6.1.4.1.2.6.3.1           6          58916864
2     1.3.6.1.4.1.2.6.3.1           6          58916865
2     1.3.6.1.4.1.2.6.3.1           6          58916866
2     1.3.6.1.4.1.2.6.3.1           6          58916867
1     1.3.6.1.4.1.2.6.3.1           6          58916965
6     1.3.6.1.4.1.2.6.3.1           6          58982401
3     1.3.6.1.4.1.2.3.1.2.1.1.3     4          0
Total Traps: 23

=== Event Statistics for hour 15 =============================

Count Enterprise_ID                 Generic_ID Specific_ID
1     1.3.6.1.4.1.2.6.3.1           6          1357
1     1.3.6.1.4.1.2.6.3.1           6          1358
1     1.3.6.1.4.1.2.6.3.1           6          58785793
1     1.3.6.1.4.1.2.6.3.1           6          58785795
5     1.3.6.1.4.1.2.6.3.1           6          58916865
6     1.3.6.1.4.1.2.6.3.1           6          58916867
2     1.3.6.1.4.1.2.6.3.1           6          58982401
2     1.3.6.1.4.1.2.6.3.1           6          1357
Total Traps: 19

=== Event Statistics for hour 16 =============================

Count Enterprise_ID                 Generic_ID Specific_ID
2     1.3.6.1.4.1.2.6.3.1           6          1358
1     1.3.6.1.4.1.2.6.3.1           6          50790404
1     1.3.6.1.4.1.2.6.3.1           6          58916864
8     1.3.6.1.4.1.2.6.3.1           6          58916865
1     1.3.6.1.4.1.2.6.3.1           6          58916866
8     1.3.6.1.4.1.2.6.3.1           6          58916867
1     1.3.6.1.4.1.2.6.3.1           6          58916868
1     1.3.6.1.4.1.2.6.3.1           6          58916869
2     1.3.6.1.4.1.2.6.3.1           6          59179056
2     1.3.6.1.4.1.2.6.3.1           6          59179057
1     1.3.6.1.4.1.2.6.3.1           6          58916864
Total Traps: 28
```

Figure 167. Shell Script for Analyzing the Ruleset-Created Trap Log

Use of the logsort shell script only gives us a first step in the direction of event stream analysis. However, we believe that it provides a sound basis for a more thorough approach.

Chapter 9. Agent Policy Manager (APM)

Agent Policy Manager is a feature of NetView for AIX Version 4 that simplifies the administration of Systems Monitor agents. In this project we did not work a great deal with APM, so this section merely describes its capabilities and shows a simple worked example of its capability. A more detailed discussion of APM will be published during 1Q96 in *IBM Systems Monitor: Anatomy of a Smart Agent*, SG24-4398 (the previous version of this book is numbered GG24-4398).

9.1 Systems Monitor Agents

Systems Monitor is a family of SNMP agents for UNIX systems. The family currently contains the following six members:

1. AIX Systems Monitor Systems Information Agent (SIA), Version 2 Release 2
2. AIX Systems Monitor Mid-Level Manager (MLM), Version 2 Release 2
3. AIX Systems Monitor System Level Manager (SLM), Version 2 Release 2
4. HP-UX Systems Monitor Agent, Version 1 Release 2
5. Sun Systems Monitor Agent, Version 1 Release 2
6. NCR Systems Monitor Agent, Version 1 Release 2

The functions provided by these agent can broadly be subdivided into the following categories:

Instrumentation	An extended MIB that provides information about many different aspects of a UNIX system, such as CPU, disk and paging performance, active processes and users. On AIX this function is provided by the SIA. On HP, Sun and NCR machines it is provided by the single Systems Monitor agent.
Extensibility	The Systems Monitor command table is a MIB extension that allows any UNIX command to be executed as a result of an SNMP GET request. This is also provided by the SIA for AIX nodes and by the Systems Monitor Version 1 agent for the other UNIX varieties.
File Monitoring	This is a regular monitoring process that can examine files on the agent system and generate SNMP traps if it detects changes to the files, or specified error text within them. The File Monitor table is only available on the AIX SIA.
MIB Threshold Polling	This function provides regular polling of MIB variables. It is provided by the AIX MLM and also by the Version 1 agents for the other UNIX types. The MIB variable being polled can be on any SNMP-capable agent node, so you can think of this function as a way of removing some load from NetView for AIX, by distributing it to multiple mid-level managers. The Threshold Table is also provided by the AIX SLM agent, but in that case it can only poll MIB variables on the same (SLM) machine.

Trap Filtering This function is provided by the AIX MLM and by the Version 1 agents for the other UNIX types. It allows traps sent from any SNMP agent to be compared against predefined criteria and either blocked or forwarded on to NetView for AIX.

The filter table is also provided by the AIX SLM agent, but in that case it can only receive traps from the same (SLM) machine.

Status Polling This function allows Systems Monitor to perform regular node status polling on behalf of NetView for AIX. This is an ICMP echo (ping) to all the interfaces of a node that is used to check for availability. Status Polling is only provided by the AIX MLM. Placing the status polling function on a mid-level manager reduces load on NetView for AIX and the network.

There are other Systems Monitor functions that support these main capabilities. If you want to understand Systems Monitor in more detail you should refer to *IBM Systems Monitor: Anatomy of a Smart Agent*, GG24-4398 (this book will be available in a revised edition during 1Q96, renumbered as SG24-4398).

9.2 Administering Systems Monitor

Many of the Systems Monitor functions require configuration updates to be made on the agent nodes. To illustrate the kind of setup changes that are needed, consider the hierarchy of agents shown in Figure 168 on page 203.

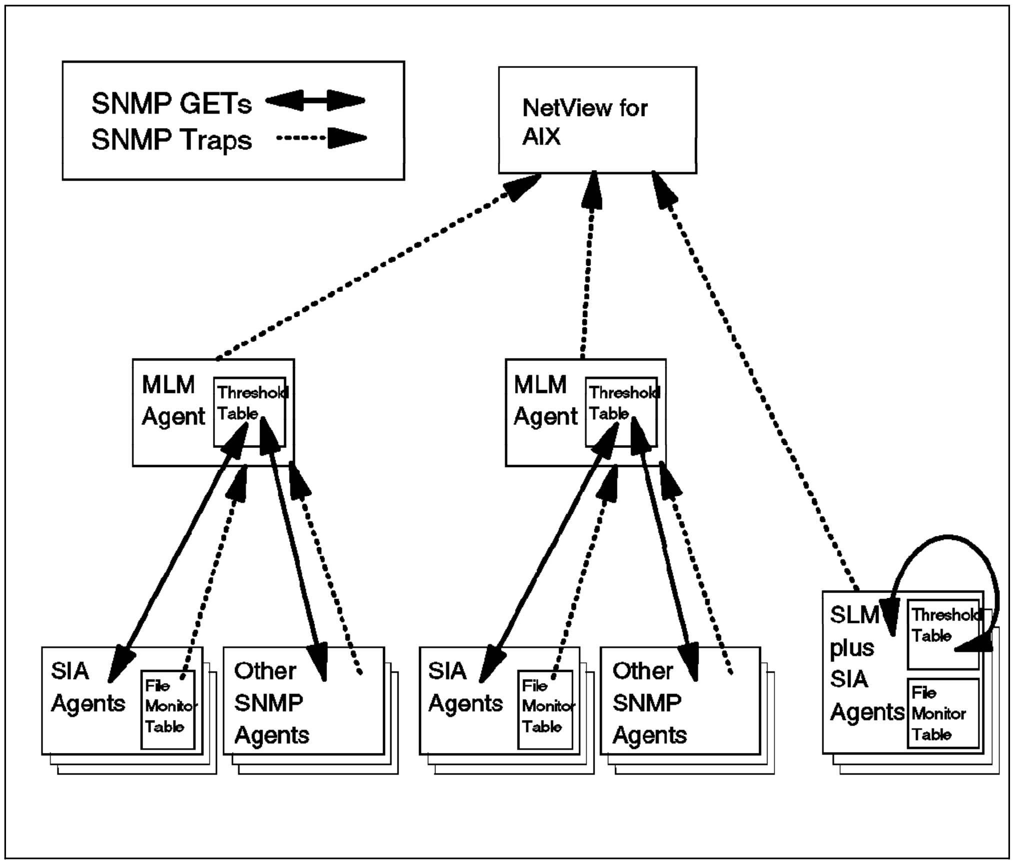


Figure 168. Hierarchy of Systems Monitor Agents

In this case we can see that there are two MLMs, each responsible for polling a group of agents, some of them SIAs and some of them regular SNMP agents. There are also several SLMs in the network. NetView automatically takes care of configuring the status polling function, but the threshold and file monitor tables have to be manually maintained. Setting up these tables would involve the following steps:

- Configure the SNMP community names and trap destinations on each of the MLM and SIA nodes.
- Create file monitor configurations on each SIA (or create them once and copy the configuration file to each of the other SIA nodes).
- Create alias entries on the MLMs, listing the agents under its control, and grouping them if required.
- Create threshold table configurations on the MLMs and SLMs to poll for the required MIB variables, associating them with the node aliases where applicable.

All of this can be executed from the central Systems Monitor configuration interface, but clearly it is a rather repetitive task.

9.3 How APM Helps

The agent policy manager function of NetView for AIX provides these three enhancements for the administration and use of Systems Monitor agents:

1. A single interface for configuring file monitor and threshold tables across multiple agent nodes.
2. Automatic update of Systems Monitor tables as nodes join and leave the network.
3. Display of icons to represent the status of threshold and file monitor table entries in NetView for AIX submaps.

The first two enhancements rely on the collection facility. To illustrate how it works, consider again the hierarchy shown in Figure 168 on page 203. Suppose we want to use APM to define a file monitor entry and a threshold entry for all of the SIA nodes in the diagram. First we would create a collection containing all the nodes. This is easy, since there is a flag in the object database which indicates if a node is an SIA or not (in fact we could use the collection called siaNodes which is automatically created by NetView for AIX).

To create the file monitor table entry we would define the information we wanted to monitor and then tell APM to distribute the update. Since the file monitor table exists on each SIA, it would update every agent in the collection. If a new SIA node appeared in the network, APM would add our monitor to it automatically, as soon as the node was discovered.

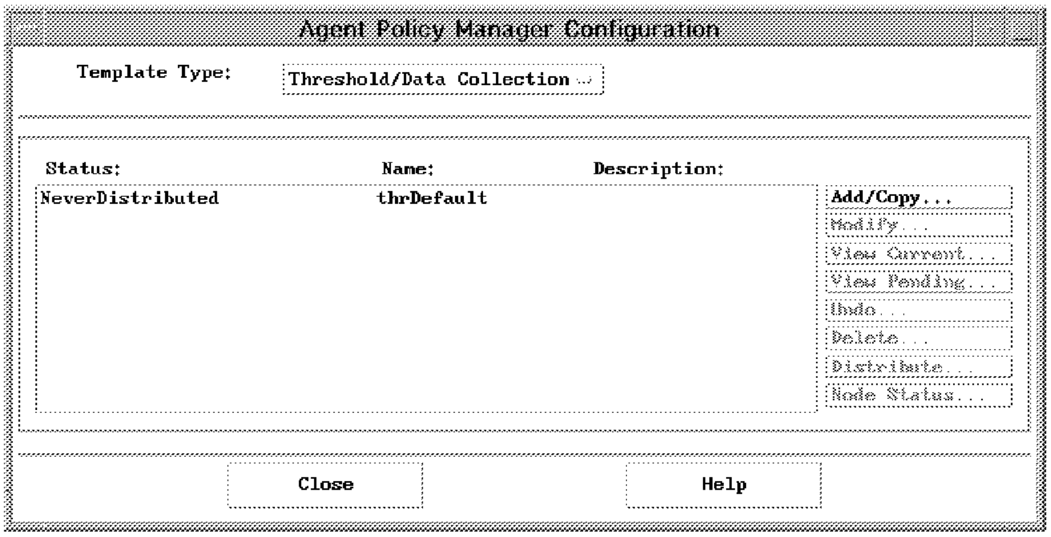
To create the threshold monitor we would go through exactly the same procedure; creating the entry and then distributing it. However, in this case the monitoring is performed by MLM or SLM nodes, so instead of sending an update to all of the nodes in the collection, APM would only update the SLMs and MLMs. NetView for AIX creates node collections containing the list of nodes managed by a given MLM. As before, if a new SIA or SLM node appeared in the network the threshold tables would automatically be updated to include it.

9.4 An APM Example

The best way to describe the function of APM is to show an example of its use. In this example we show the creation of a threshold monitor for a collection of nodes. We use the collection called Servers that we created in 7.1.2, “Manipulating Event Severity on the Basis of Node Importance” on page 175 for this. The policy that we want to implement is to check regularly to make sure that all the nodes in the Servers collection have the SNA subsystem active.

The process to implement this monitor is as follows:

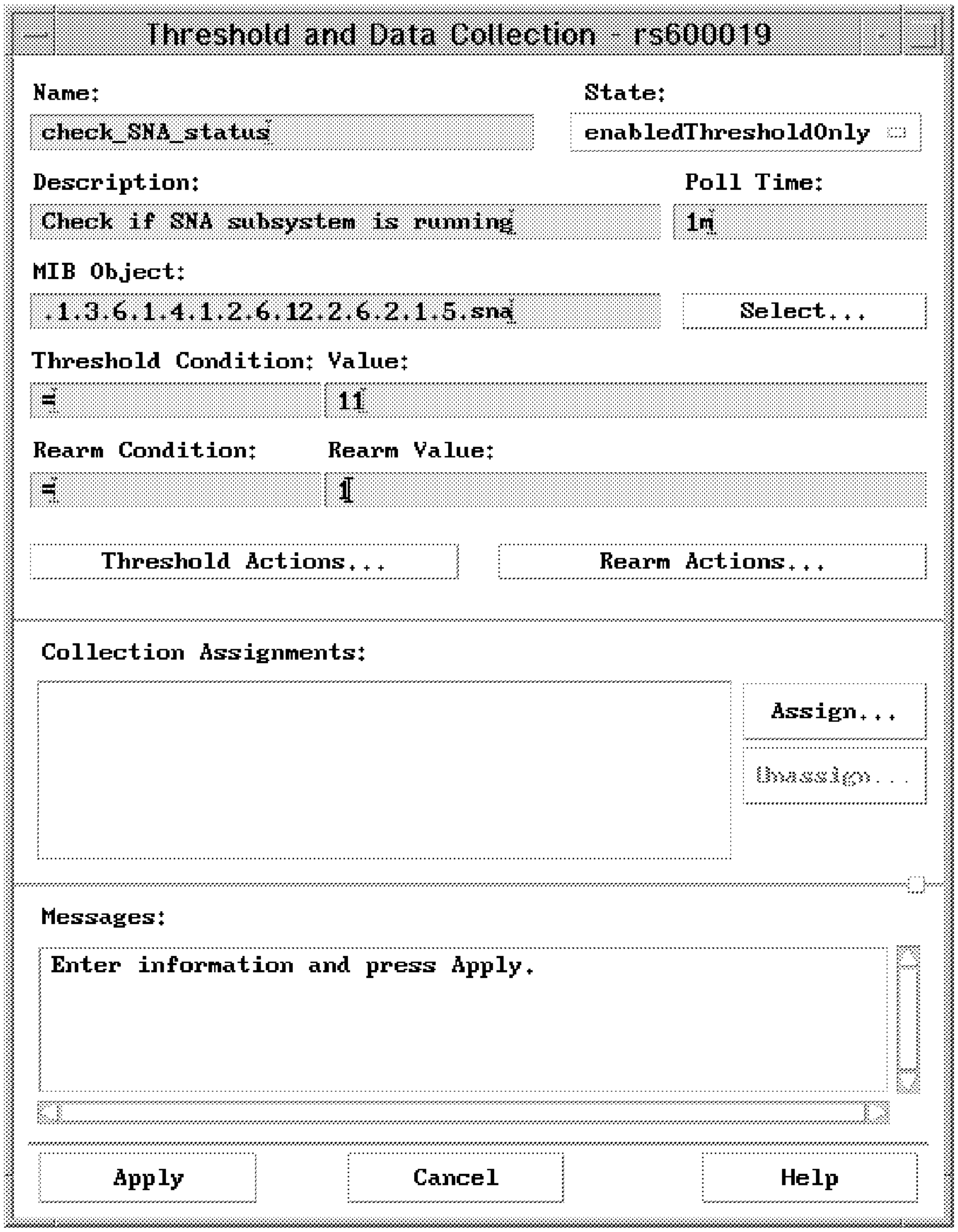
1. Select **Agent Policy Manager Configuration** from the Tools menu bar entry. This displays the panel shown in Figure 169 on page 205.



```
Status:                Name:           Description:
NeverDistributed       thrDefault
```

Figure 169. The APM Configuration Panel

2. To create a threshold table entry, click on **Add** to see the configuration panel shown in Figure 170 on page 206. If you know Systems Monitor you will recognize that this is very similar to the panel that you would use to configure a threshold table entry on a single node.



Threshold and Data Collection - rs600019

Name: check_SNA_status

State: enabledThresholdOnly

Description: Check if SNA subsystem is running

Poll Time: 1m

MIB Object: .1.3.6.1.4.1.2.6.12.2.6.2.1.5.sna

Threshold Condition: =

Value: 11

Rearm Condition: =

Rearm Value: 1

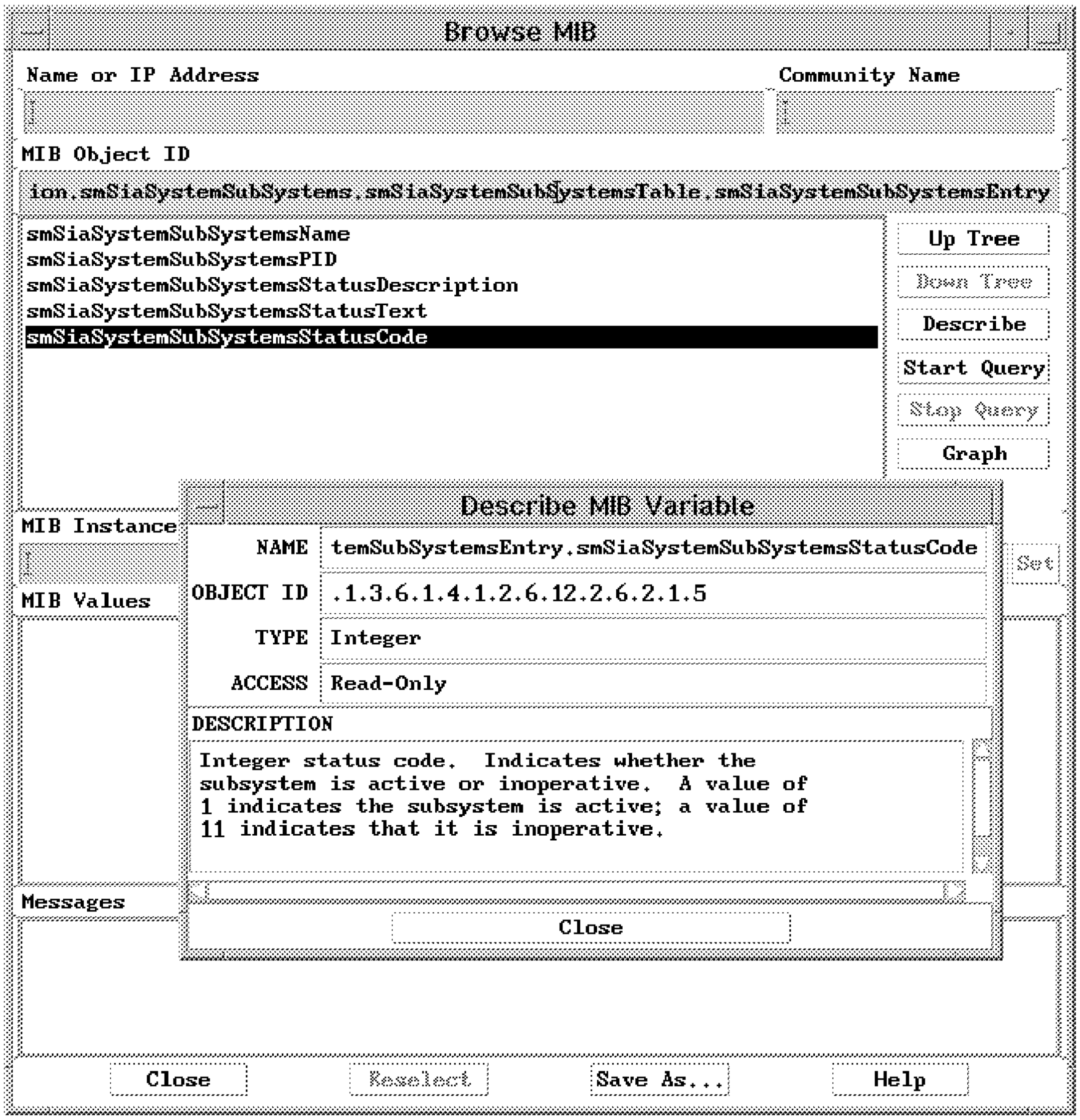
Collection Assignments:

Messages:

Enter information and press Apply.

Figure 170. Configuration Panel for a Threshold Monitor

3. Next, fill in the details if the monitor. As you can see in Figure 170, we are polling at one minute intervals and we will consider a threshold to be breached when the selected MIB variable is equal to 11. The particular MIB object we are polling is part of the instrumentation provided by the Systems Monitor SIA. You can find the object ID and description by clicking on **Select...** which takes you into the MIB Browser application. Figure 171 on page 207 shows the MIB Browser dialog, with the description of the object which we want to check against a threshold, in this case the subsystem status code. on.



ion.smSiaSystemSubSystems.smSiaSystemSubSystemsTable.smSiaSystemSubSystemsEntry

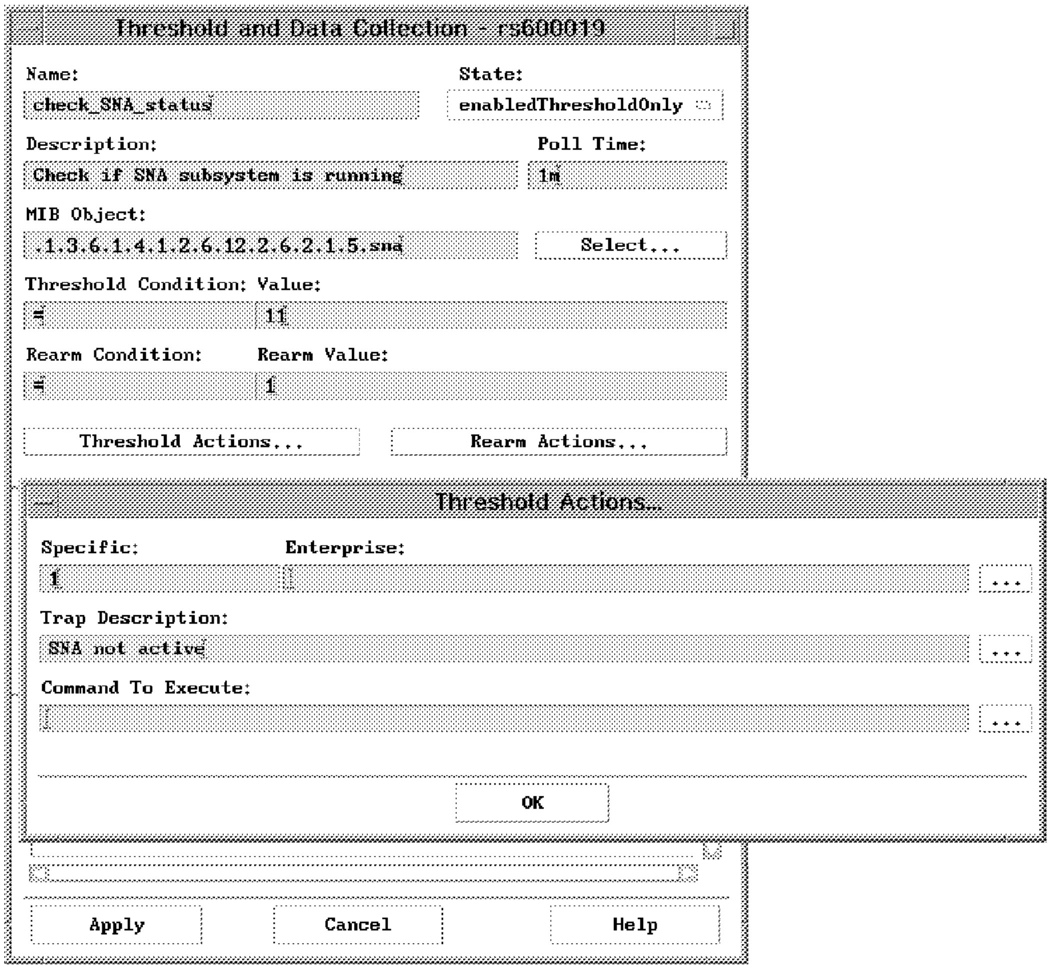
smSiaSystemSubSystemsName
smSiaSystemSubSystemsPID
smSiaSystemSubSystemsStatusDescription
smSiaSystemSubSystemsStatusText
smSiaSystemSubSystemsStatusCode

NAME	temSubSystemsEntry.smSiaSystemSubSystemsStatusCode
OBJECT ID	.1.3.6.1.4.1.2.6.12.2.6.2.1.5
TYPE	Integer
ACCESS	Read-Only

Integer status code. Indicates whether the subsystem is active or inoperative. A value of 1 indicates the subsystem is active; a value of 11 indicates that it is inoperative.

Figure 171. Using the MIB Browser to Identify MIB Object IDs. You can save effort by performing a cut-and-paste operation to copy the Object ID from the Describe MIB Variable pop-up into the configuration dialog.

4. The final part of defining the threshold table entry is to set the action to be performed when the threshold is breached or re-armed. Click on **Threshold Actions** to get the panel shown in Figure 172 on page 208. In this case we will send an SNMP trap to warn that the SNA subsystem is not active.


Threshold and Data Collection - rs600019

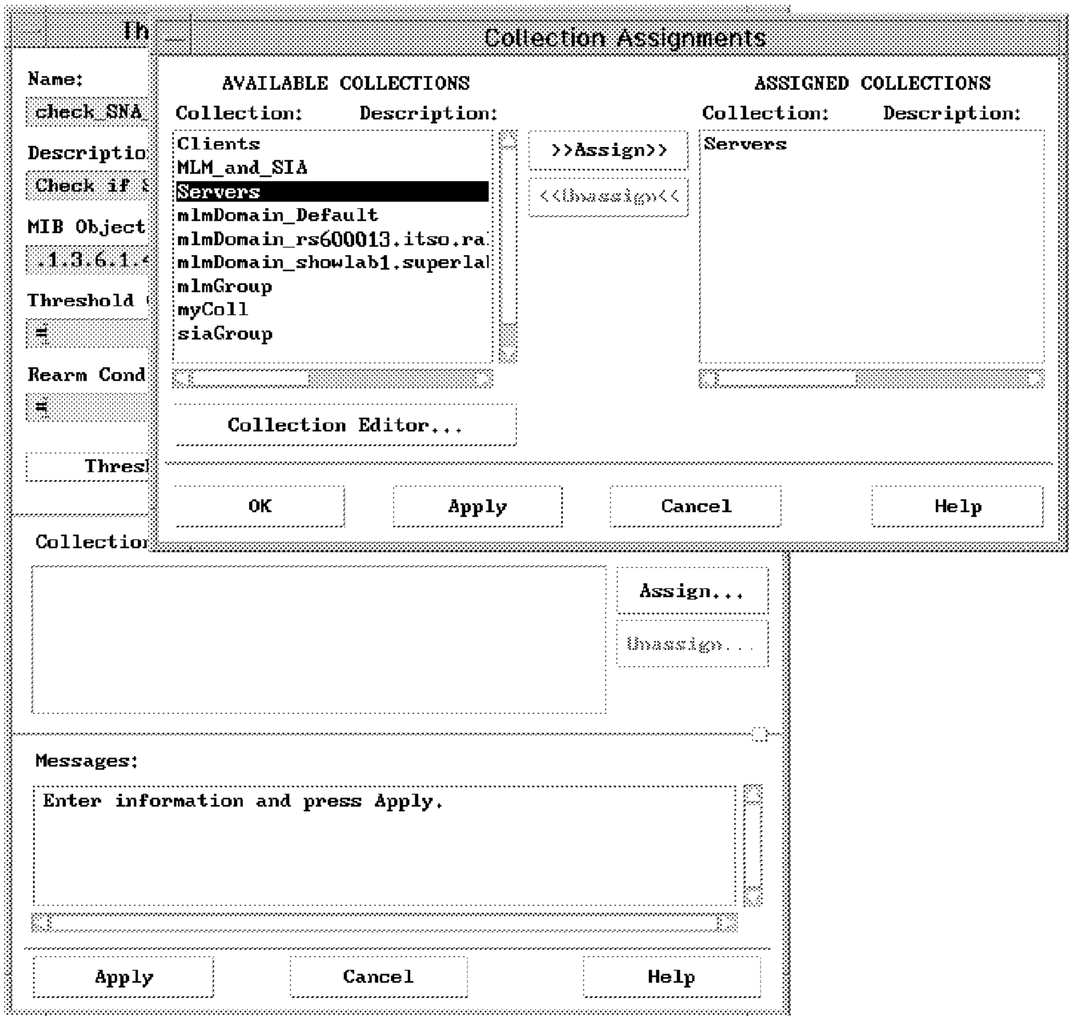
Name: check_SNA_status
State: enabledThresholdOnly
Description: Check if SNA subsystem is running
Poll Time: 1m
MIB Object: .1.3.6.1.4.1.2.6.12.2.6.2.1.5.sna
Threshold Condition: =
Value: 11
Rearm Condition: =
Rearm Value: 1

Threshold Actions...

Specific: 1
Enterprise:
Trap Description: SNA not active
Command To Execute:

Figure 172. Defining Threshold Actions

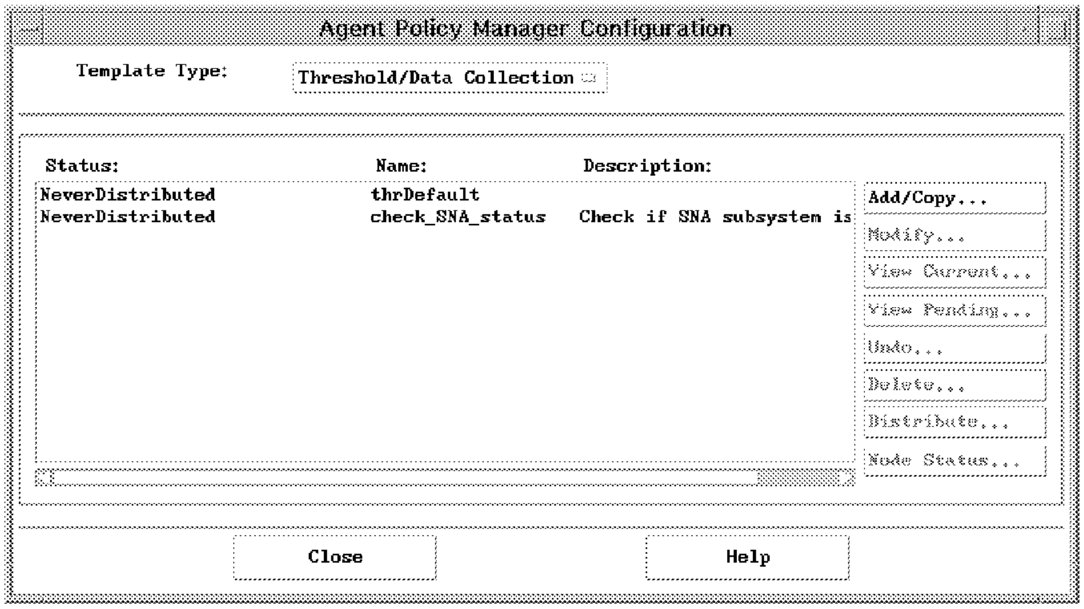
5. Having defined the threshold table entry, we next need to define which nodes it is to apply to. Click on **Assign...** to see the panel shown in Figure 173 on page 209. In this dialog you can select the node collections that you want to monitor. In our case we just selected one collection, **Servers**.



Enter information and press Apply.

Figure 173. Assigning a Collection to the Threshold Monitor

6. You can now select **Apply** and then **Cancel** to return to the main configuration panel. The newly defined entry will appear on the list, with a status of NeverDistributed (see Figure 174 on page 210).


Agent Policy Manager Configuration

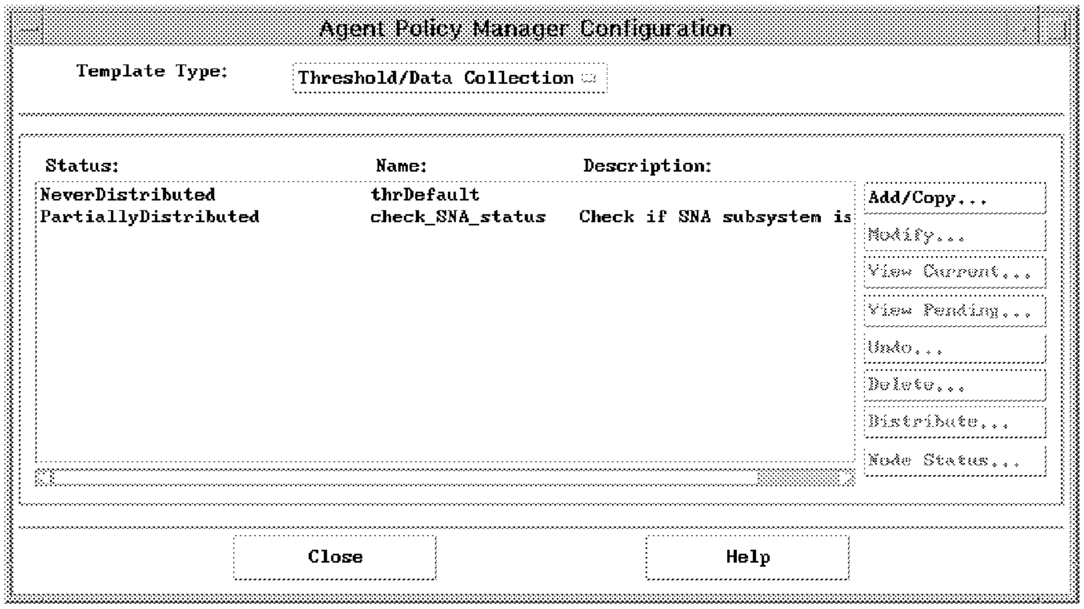
Template Type: Threshold/Data Collection

Status:	Name:	Description:
NeverDistributed	thrDefault	
NeverDistributed	check_SNA_status	Check if SNA subsystem is

Figure 174. Threshold Entry Defined but not yet Distributed

7. To distribute the new configuration, select it from the list and click on **Distribute** and then **Start**. At this point APM will determine which Systems Monitor nodes to send the update request to and perform the update. In our case we are defining a threshold table entry, so the distribution will be to MLM and SLM nodes only.

   When we tried the distribution for our example configuration, we received an error and the status changed to PartiallyDistributed (see Figure 175).

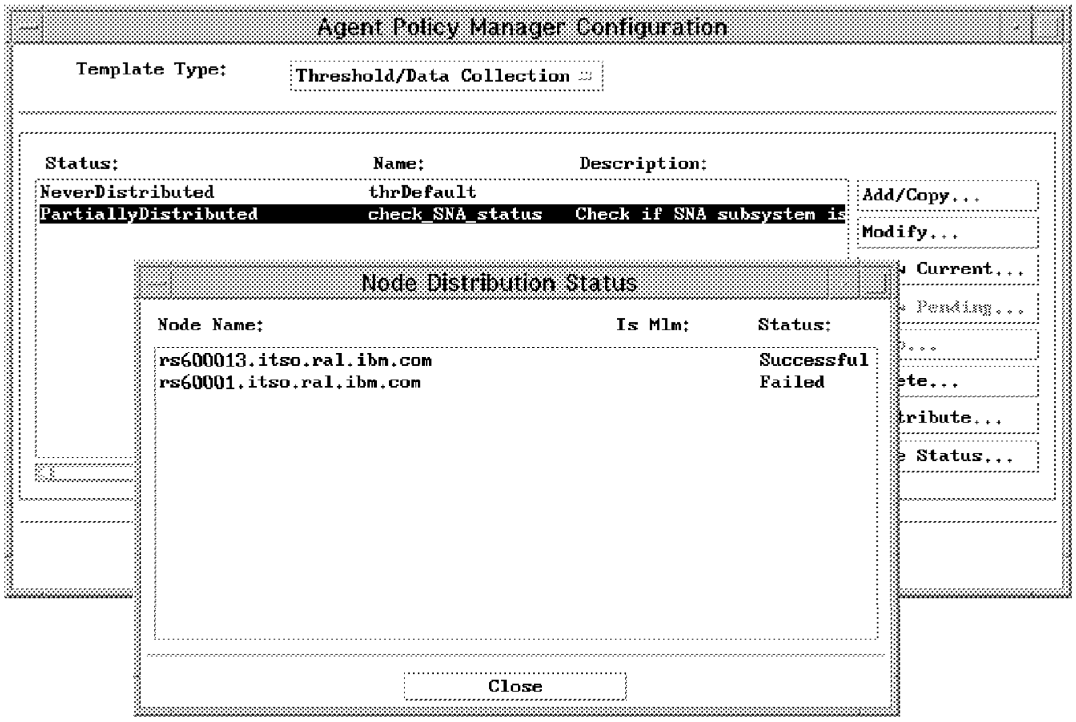

Agent Policy Manager Configuration

Template Type: Threshold/Data Collection

Status:	Name:	Description:
NeverDistributed	thrDefault	
PartiallyDistributed	check_SNA_status	Check if SNA subsystem is

Figure 175. Result of a Failed Distribution Attempt

There are error message generated during the distribution process, but if you need to check up which target nodes were successfully updated you can find out by clicking on **Node Status** (see Figure 176 on page 211).



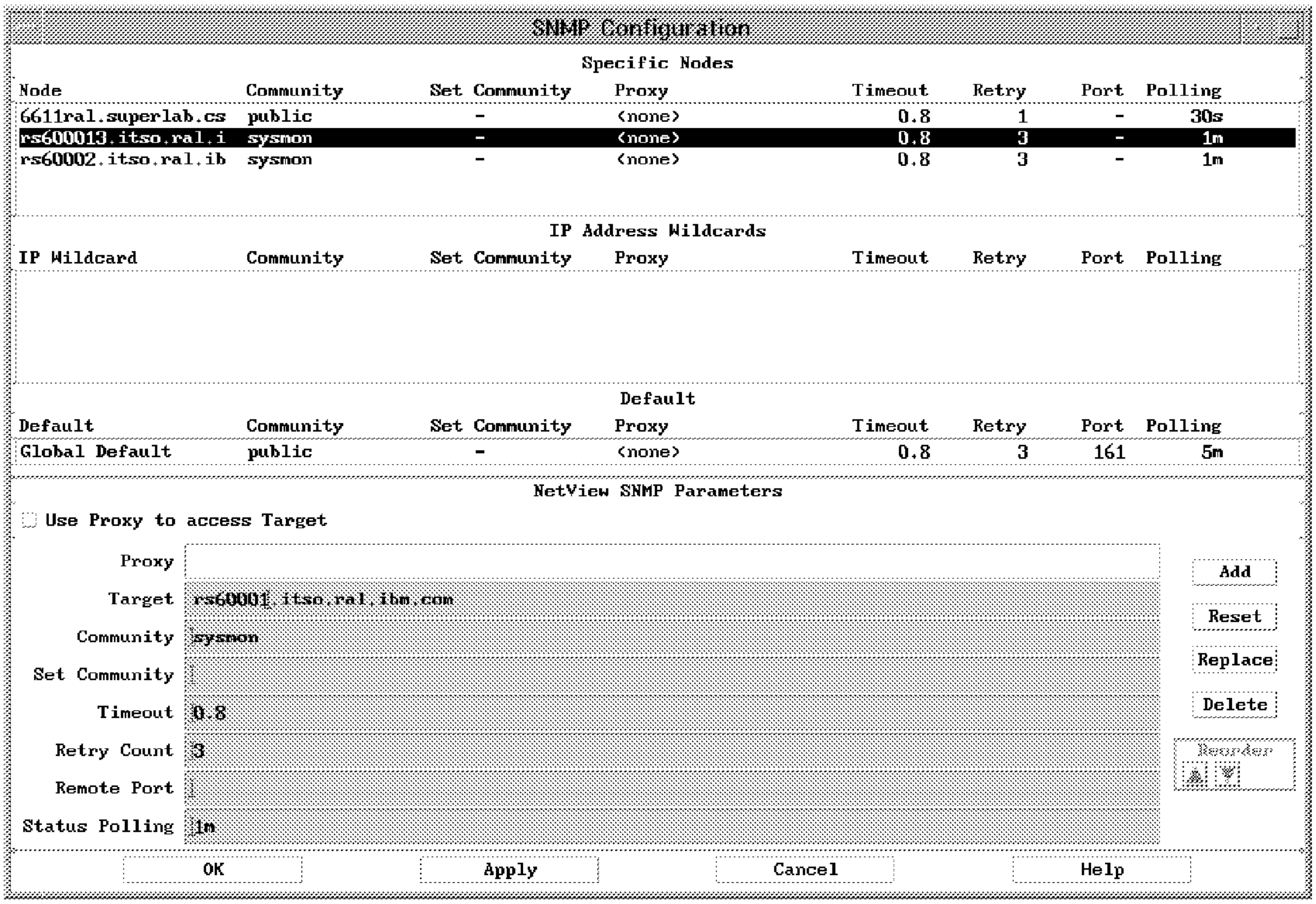
```
Status:                 Name:              Description:
NeverDistributed        thrDefault
PartiallyDistributed    check_SNA_status   Check if SNA subsystem is
```

```
Node Name:                          Is Mlm:    Status:
rs600013.itso.ral.ibm.com                      Successful
rs60001.itso.ral.ibm.com                       Failed
```

Figure 176. Listing the Distribution Status

Figure 176 shows us the nodes that APM has decided to update to monitor the Servers collection. If you compare the nodes listed (rs60001 and rs600013) with the list of nodes in the collection (see Figure 148 on page 177) you will see that although rs60001 *is* in the collection, rs600013 is not. The reason for this is that rs600013 has the MLM agent running, and it will perform the threshold monitoring for most of the nodes in the collection. rs60001 is an exception to this because it has the SLM agent installed, so it can perform its own threshold monitoring locally.

8. Next we have to determine why the distribution failed for rs60001. One key item to remember when using Systems Monitor agents is that the tables are configured using SNMP SET requests. This means that NetView for AIX has to use a community name with read/write access on the agent node. In the case of our distribution failure, we had not defined such a community name for rs60001, which is why the problem occurred. You define the community names that NetView for AIX will use by selecting Options ==> SNMP Configuration from the menu bar. Figure 177 on page 212 shows this dialog.



SNMP Configuration

Specific Nodes

```
Node               Community   Set Community  Proxy        Timeout   Retry   Port  Polling
6611ral.superlab.cs public        -           <none>         0.8       1      -      30s
rs600013.itso.ral.i sysmon        -           <none>         0.8       3      -      1m
rs60002.itso.ral.ib sysmon        -           <none>         0.8       3      -      1m
```

IP Address Wildcards

```
IP Wildcard        Community   Set Community  Proxy        Timeout   Retry   Port  Polling
```

Default

```
Default            Community   Set Community  Proxy        Timeout   Retry   Port  Polling
Global Default     public         -           <none>         0.8       3     161     5m
```

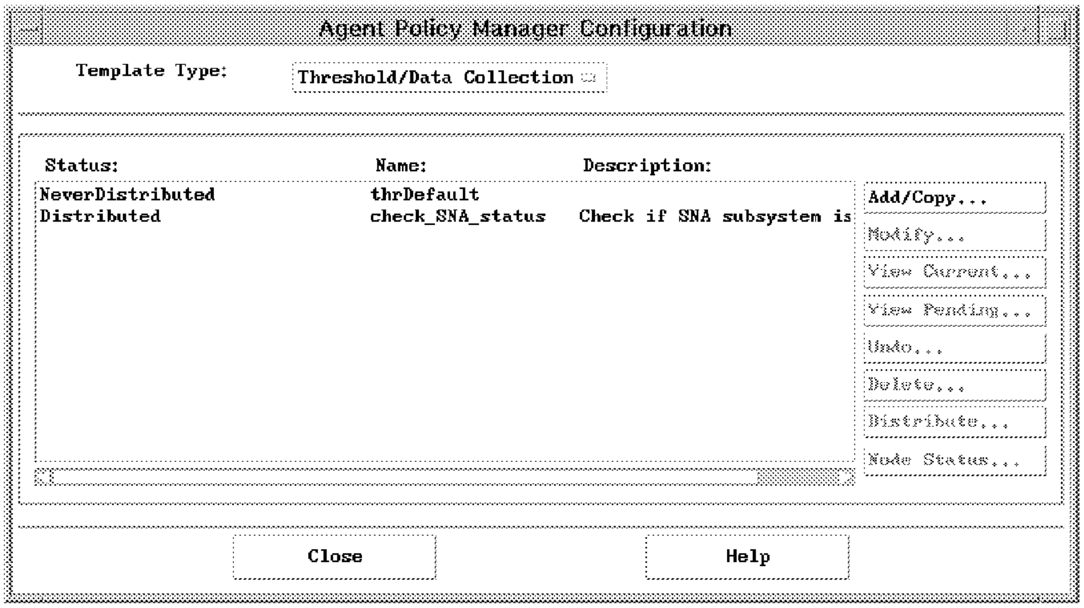
NetView SNMP Parameters

☐ Use Proxy to access Target

Proxy	
Target	rs60001.itso.ral.ibm.com
Community	sysmon
Set Community	
Timeout	0.8
Retry Count	3
Remote Port	
Status Polling	1m

Figure 177. Defining Community Name and Status Polling Interval

9. Finally we can retry the distribution from the APM configuration panel, in the same way as before. This time the update for rs60001 is successful, as shown in Figure 178.



Agent Policy Manager Configuration

Template Type: Threshold/Data Collection

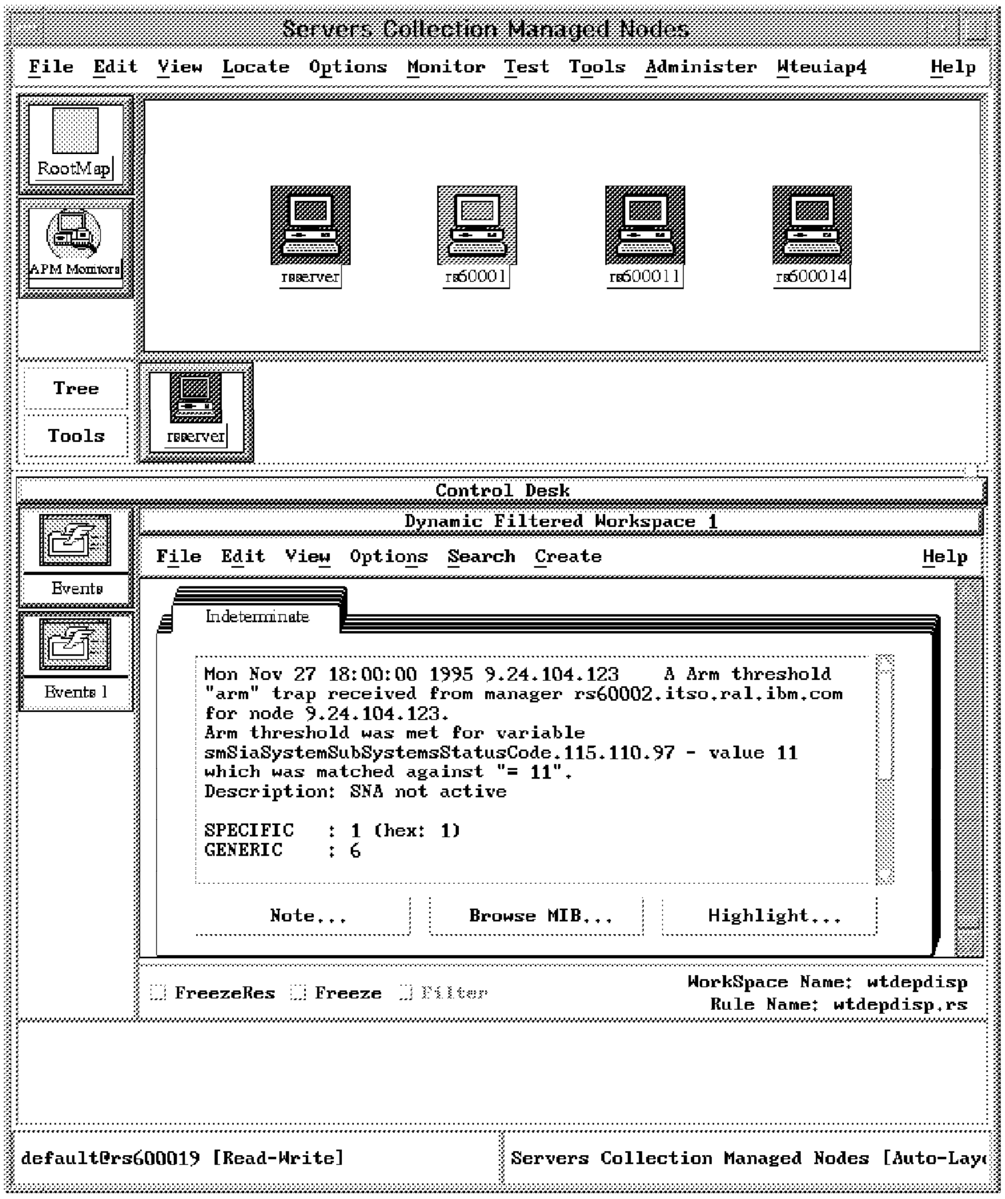
```
Status:              Name:              Description:
NeverDistributed     thrDefault
Distributed          check_SNA_status   Check if SNA subsystem is
```

Figure 178. Successful Distribution of the Threshold Example

9.4.1 Results from APM Monitors

APM creates standard Systems Monitor threshold and file monitors. These can provide warnings by means of SNMP traps and they can also optionally execute automated commands when they are triggered or re-armed. However, APM adds an additional display function that represents the status of the monitor graphically.

Using our previous threshold monitor example again, we stopped the SNA subsystem on one of the nodes in the Servers collection. The result is that an event card appears and the symbol representing the node changes color to red in the APM monitors submap (seeFigure 179).


Servers Collection Managed Nodes
File Edit View Locate Options Monitor Test Tools Administer Wteuiap4 Help
RootMap
APM Monitors
rsserver
rs60001
rs600011
rs600014
Tree
Tools
rsserver
Control Desk
Dynamic Filtered Workspace 1
File Edit View Options Search Create Help
Events
Events 1
Indeterminate
Note...
Browse MIB...
Highlight...
FreezeRes
Freeze
Filter

```
Mon Nov 27 18:00:00 1995 9.24.104.123    A Arm threshold
"arm" trap received from manager rs60002.itso.ral.ibm.com
for node 9.24.104.123.
Arm threshold was met for variable
smSiaSystemSubSystemsStatusCode.115.110.97 - value 11
which was matched against "= 11".
Description: SNA not active

SPECIFIC   : 1 (hex: 1)
GENERIC    : 6
```

```
WorkSpace Name: wtdepdisp
Rule Name: wtdepdisp.rs
default@rs600019 [Read-Write]
Servers Collection Managed Nodes [Auto-Lay
```

Figure 179. Result of Inoperative SNA Subsystem Being Detected

There is also an additional symbol added to the node submap for the affected node. Figure 180 on page 214 shows the symbol for our SNA subsystem status monitor, along with the IP interface symbols that are always present.

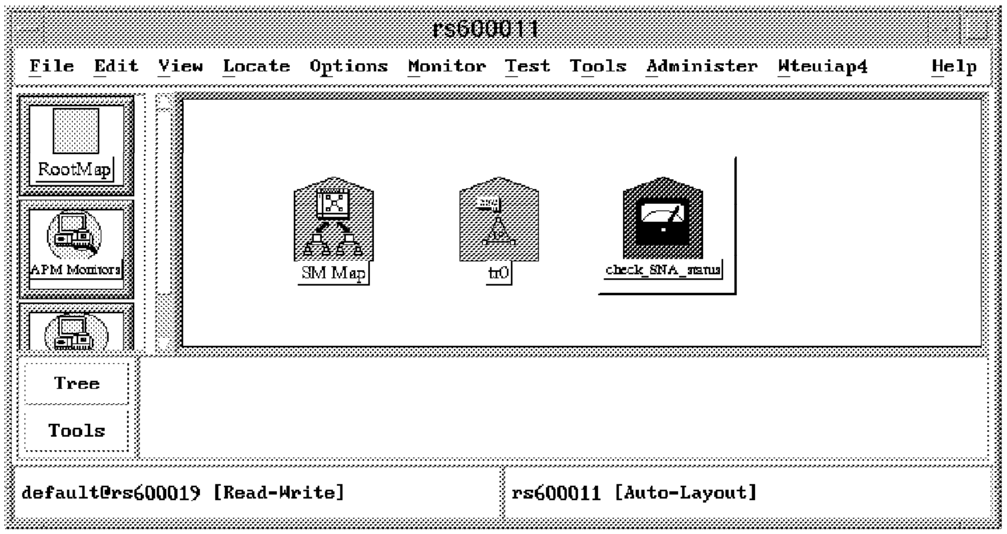

rs600011
File Edit View Locate Options Monitor Test Tools Administer Wteuiap4 Help
RootMap
APM Monitors
SM Map
tr0
check_SNA_status
Tree
Tools
default@rs600019 [Read-Write]
rs600011 [Auto-Layout]

Figure 180. Symbol Representing a Threshold Monitor

9.5 Activating APM

There are two main components of the Agent Policy Manager:

- A daemon, C5d, which maintains the Systems Monitor tables.
- A background process, C5eui, which updates the APM submaps and status symbols. There is one copy of C5eui running for each NetView for AIX user.

The APM processes are not started by default when you first install NetView for AIX Version 4. To cause them to start up you must register the C5d daemon, by selecting Configure==>Set Options for Daemons==> Set Options for Agent Policy Manager from the NetView for AIX SMIT menu. You do not need to change any of the default parameters, just press Enter to perform the registration process.

Chapter 10. NetView for AIX Open Topology

The material in this chapter has been extracted from *Examples of Using NetView for AIX*, GG24-4327 and updated information has been slightly modified as a result of testing on NetView for AIX Version 4.

10.1 Introduction

NetView for AIX provides APIs to allow you to integrate programs that manage resources from different protocols.

Some examples of this might be:

1. To allow the user to select a new action from a selected object
2. An application to manage a new network protocol
3. To allow an application to react to network events

In this way, NetView for AIX may be used as a platform to extend support beyond the base capability to manage IP nodes and SNMP devices.

One key API is the OVw API, which gives a program the ability to directly modify submaps and the object information underlying them. However, if you want to manage and integrate networks that use other protocols, NetView for AIX provides you with an alternative approach (*Open Topology*).

Open Topology has several advantages over using the OVw API directly:

1. Simplification

   All of the work to create the submaps and linkages between them is handled by NetView for AIX, so user code may be much less complex.

2. Integration and correlation

   A standard part of the open topology function is the ability to identify situations where one object appears as a symbol in more than one network protocol, and provide linkage between them.

3. Protocol switching

   You can select the list of protocols associated with the object, and then switch to the submap representing the required protocol. This allows you to display the object in the context of the protocol that you are investigating.

   For example, switch between the IP and SNA views of a PS/2.

4. Integration with the control desk event cards

   When there is an event card displayed, then the source of this event can be highlighted. This function is standard, if the application uses NetView for AIX Open Topology.

5. Propagation of status between protocols

   NetView for AIX Open Topology allows for a status change in a protocol symbol to be automatically propagated to the object that is hosting the protocol.

10.2 Open Topology Components

Figure 181 on page 217 shows the elements that make up the IP Topology and the Open Topology parts of NetView for AIX. Notice the symmetry between the two halves of the diagram; for each function that specifically deals with IP Topology, there is an equivalent non-IP function. We will examine the roles played by each of these component pairs, as follows:

netmon/Application code The netmon daemon performs discovery for the IP network, and then polls for status. An application monitoring a non-IP protocol has to provide the same function. The mechanism used for monitoring may be specific to the protocol, or it may make use of NetView for AIX facilities. A good example of the latter would be LAN Management Utilities/6000, which monitors PS-based client/server environments using an SNMP proxy agent.

iptopmd/gtmd These two daemons generate and maintain the databases that contain topology information (iptopmd for IP and gtmd for Open). Each creates its own database based on an abstract *model* of how a network is constructed.

iptopmd uses an internal IP-specific model, and builds its database from this, based on information provided by netmon. gtmd uses the IBM Open Topology model to build its database using information from specially formatted traps. Both daemons also create and maintain entries in the object database.

The Open Topology model is defined by a MIB, the source for which is found in file /usr/OV/snmp_mibs/drafts/nv6k_topo.mib. This MIB defines network elements, plus the trap formats for defining them, in a generic form.

ipmap/xxmap These two background processes are started whenever a user starts the NetView for AIX GUI. They take information from the object database and the topology databases (ipmap for IP and xxmap for Open) and convert it into submap and symbol definitions. They remain active, being responsible for maintaining symbol status. xxmap additionally provides the symbol correlation and protocol switching function previously described.

The only element of the Open Topology support that we have not mentioned is noniptopod. This daemon registers with netmon to receive traps when a new node is discovered. Using the IP node address and OID from the trap, it sends SNMP get requests for each OID in the oid_to_command file, to the node named in the trap. If it receives a valid responses, it sends the command named in the oid_to_command file to start the data collection process on the node. We will discuss this further in 10.4, “Network Discovery with Open Topology” on page 221.

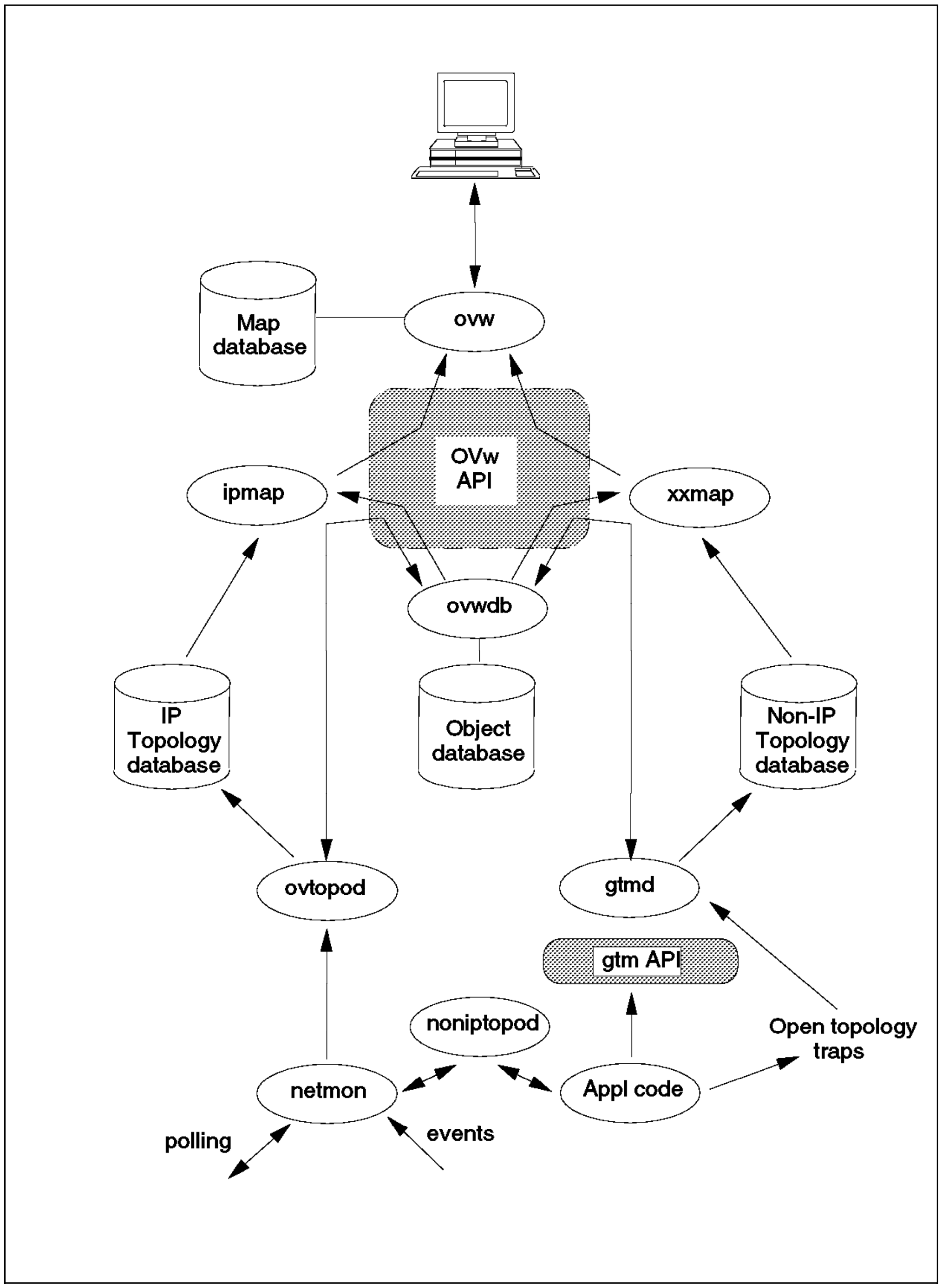


Figure 181. Components of IP and Open Topology

As you can see, all that an application has to do to create a set of submaps depicting its own network topology is to send traps to gtmd. The format of these traps, and the MIB objects encoded within them, is defined in the Open Topology MIB.

Open Topology support was further enhanced in NetView for AIX Version 3 by the addition of an API, which may be used in place of the trap interface. Some additional enhancements were made to this API in NetView for AIX Version 4, together with some improvements in the performance of the gtmd daemon. We discuss the API further in 10.6, "The Open Topology API" on page 225.

10.3 Terms and Concepts

The Open Topology model is generic; it is designed to be applicable to many different types of network topology. The terms used are therefore somewhat abstract, and are mostly borrowed from mathematical graph theory.

One of the key concepts is of a protocol ID. This is a MIB instance that is associated with each open topology object when it is created. The default values for the protocol IDs are defined as instances of the ifType OID. The mapping between the protocol object IDs and the text that defines them is in file /usr/OV/conf/oid_to_protocol. When generating a network topology, all the objects within the topology should have the same protocol ID. So if, for example, we were creating an application to map an FDDI network, we would define the objects in the network with a protocol ID of 1.3.6.1.2.1.2.2.1.3.15, where the 15 identifies the instance for FDDI.

To add confusion, when defining the protocol ID for a *Vertex* type object (see below), the protocol ID is not referred to as a MIB instance but as an integer value. These integers are defined in the open topology MIB (/usr/OV/snmp_mibs/drafts/ibm-nv6ktopo.mib). Fortunately the values defined are all identical to the instance ID of the protocol object ID previously described. In the case of FDDI, therefore, we would define vertices using a protocol ID of 15.

In the following we list the more common types of object defined by the open topology model: This is also contained in *NetView for AIX Programmers Guide,* SC31-6238, but is included here for your convenience.

Vertex A vertex is some point in space. A vertex can contain a physical or logical interface to a network. A logical interface is a protocol such as IP or SNA. Physical interfaces are hardware adapters such as token-ring or Ethernet.

Arc Arcs represent connectivity between vertices or graphs acting as vertices. An example would be a connection between two SNA PU Type 4s. This arc connection is independent of either point.

Graph A graph is a representation of a collection of vertices and the arcs connecting them. It could represent either a physical network, like a token-ring or Ethernet segment, or it could be a logical network like IP or SNA. Graphs can also be used to group resources in a network, in any way chosen by the user.

A graph can also represent a single computer node. Each of the computer components are represented by vertices, with the appropriate connections. This sort of graph is called a box graph.

Member When graphs, arcs and vertices are contained in another graph, they are said to be *members* of that graph. For example, we may have a graph representing a token-ring segment. The vertices representing the adapters in the segment, and the arcs representing the connections between a CAU and the adapters, would all be members of the segment graph.

Underlying Arc An underlying arc is an arc that represents the lower-level connectivity used by another arc. So, for example, we might have one logical connection made up of several physical links. In this case the logical connection would be an arc, and the physical links would be underlying arcs. We say that the underlying arc is *used by* the arc.

Simple Connection A simple connection represents a connection as seen by one of the end points. It can contain any information that is specific to the end-point node. For example, if we have a switched link one end may be up (ready for connection) and the other down. A simple arc representation would only show the link as down, whereas two connections representing the ends of the link could accurately depict the status.

Service Access Point A service access point is a mechanism whereby one network element provides services to other elements. For example, a node might have IP, SNA, IPX, DECnet all using one Ethernet adapter for physical transport. A vertex representing the LAN station would *provide* a SAP. Vertices in each of the network protocols would be *using* the SAP. A vertex can only use one SAP. The SAP used by the vertex represents the lower-level protocol, and each vertex can only use one lower-level protocol. A resource can provide services to many other resources through one or more SAPs. The SAP usage is represented as a table.

We will look further at how SAPs affect NetView for AIX′s depiction of a network in 10.5, "Open Topology Service Access Points" on page 222.

Figure 182 on page 220 illustrates the relationship between several of these terms. When looking at this diagram, you should imagine each of the *layers* as NetView for AIX submaps. You would get to a lower-layer submap by exploding the higher-layer object that it is a member of or used by. This is exactly how xxmap converts Open Topology database definitions into submap format.

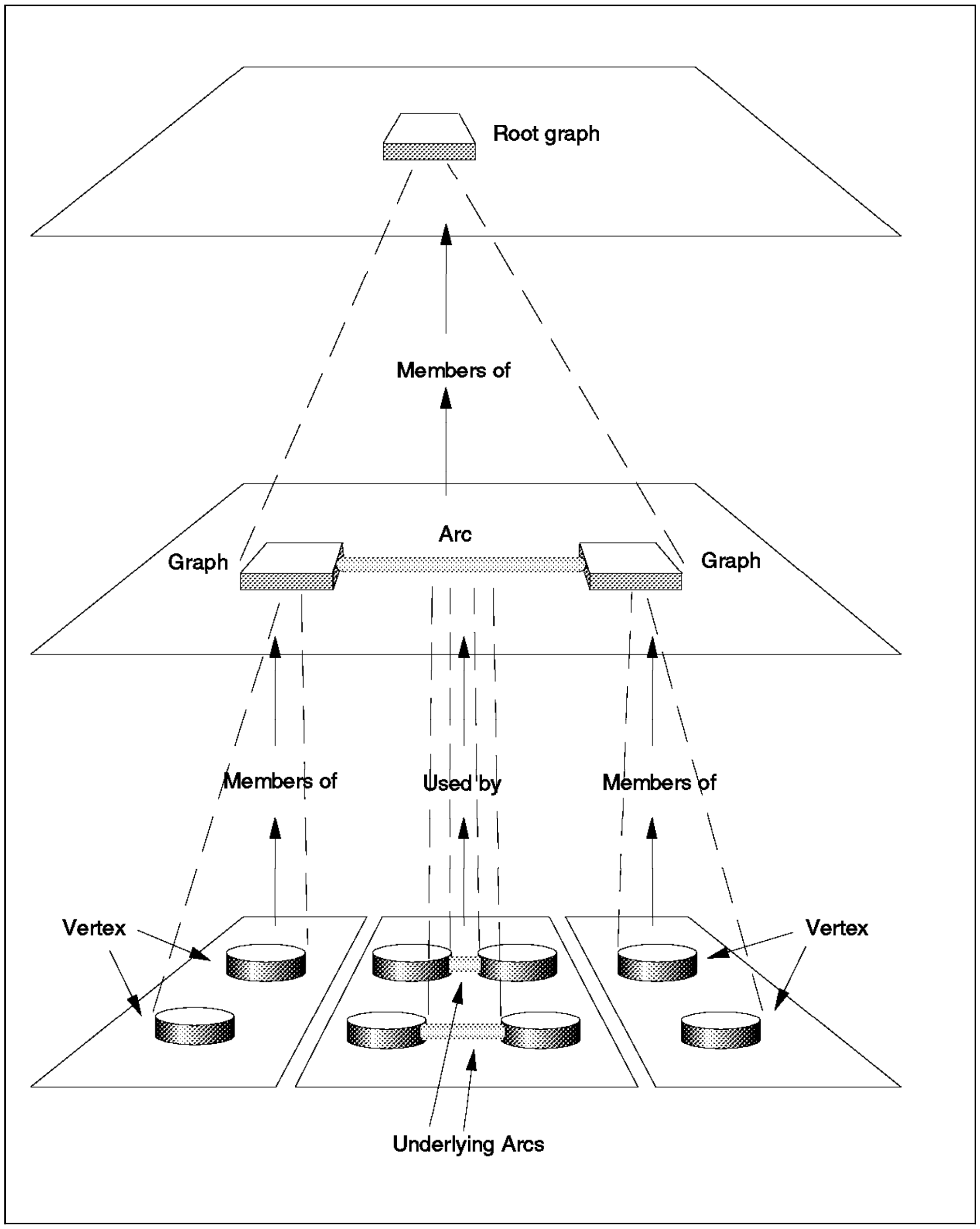


Figure 182. Some Elements of the Open Topology Model

10.3.1 Specifying Icons when Using Open Topology

Although the open topology model is mostly abstract, we want the resulting display on the NetView for AIX submap displays to be meaningful. For this reason we have to supply icon information when adding new objects. Whether we are driving open topology with traps or using the API, the icon details are passed as MIB object instances.

The file /usr/OV/conf/C/oid_to_sym provides the mapping between object instances and icons.

10.4 Network Discovery with Open Topology

NetView for AIX provides an extension to the network discovery process to allow non-IP topology applications to benefit from the discovery polling performed by netmon.

Figure 183 shows the discovery process in the context of the Open Topology support.

1. When netmon discovers a new network node, the discovery event is sent to the trapd daemon.
2. noniptopod uses event registration to allow itself to be sent copies of all these discovery events. Assuming the discovered node supports SNMP, noniptopod can use the sysObjectID retrieved from the node to extract a command from the oid_to_command file. This command would normally be a start command to start a non-IP network monitoring daemon.
3. The non-IP daemon then solicits topology information and other details from the newly discovered node.
4. The daemon then sends requests to gtmd to add the new node to the Open Topology database, and hence (via the services of xxmap) to the NetView for AIX submaps. This interaction can be via SNMP traps or via an application using the gtmd API.

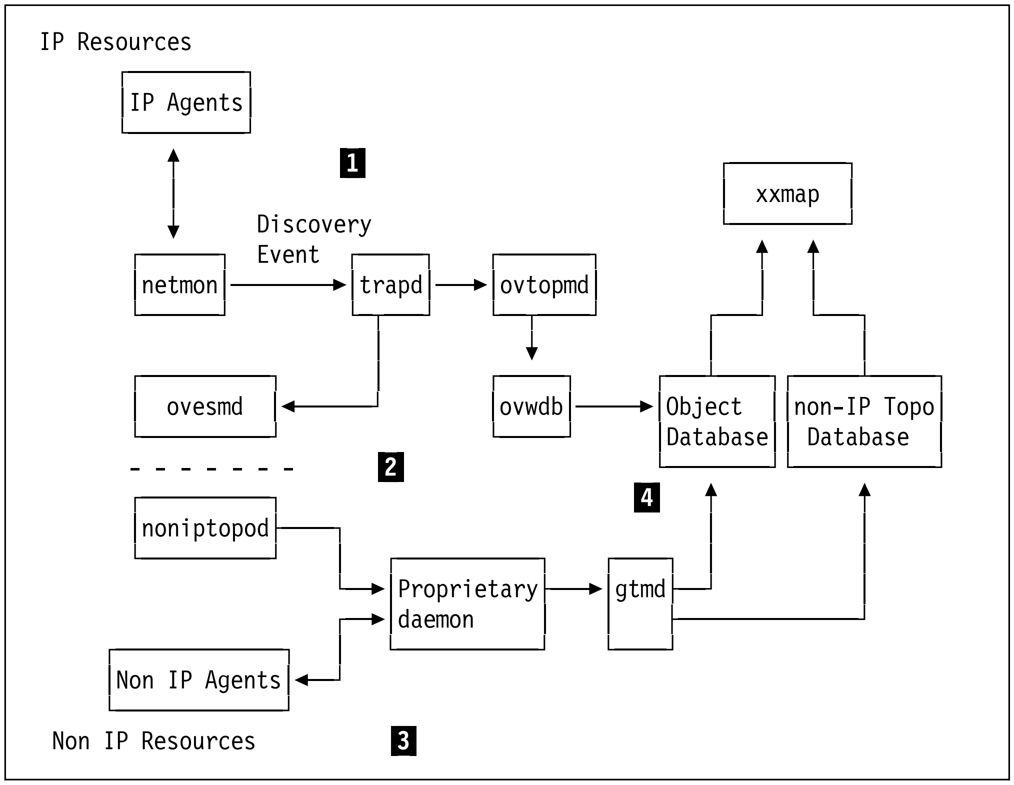


Figure 183. The Discovery Process and the Open Topology MIB

10.5 Open Topology Service Access Points

As we have discussed, the SAP table gives us a technique for declaring that a given network element *provides* a service for other network elements to *use*.

In this section we will illustrate how this affects the way that networks appear in NetView for AIX, by means of a simple example. Consider the case where we have a PS/2 with a token-ring adapter card, that is *both*:

- Physically attached to a token-ring CAU
- A node in the SNA network

We would like to present this information on NetView for AIX. Two IBM products, LNM/6000 and SNA Manager/6000, will display these network protocols, but for simplicity we will ignore the products and just look at the process from an Open Topology point of view.

We start by discussing the discovery process during which the objects are added to databases and maps.

10.5.1 The Discovery Process

During the discovery process, the applications will find the interface that is appropriate to their environment. In the case of AIX LNM/6000, NetView for AIX starts the lnm6kd daemon to talk to the LNM SNMP proxy agent. In the case of SNA Manager/6000, the SNA topology information is provided from the System/390 host in the form of preprocessed views.

Initially, there is no correlation between any of the symbols on the map, and behind the symbols, there are two separate objects.

After this initial discovery, the NetView for AIX object database will contain the following objects, in descending hierarchical order:

1. SNA (representing The SNA Network)
2. LAN (representing The LAN)
3. WS-SNA (the workstation, from an SNA point of view)
4. WS-LAN (the workstation, from a LAN physical point of view)
5. PU (the SNA physical unit for the workstation)
6. MAC Address (the LAN interface card)

The end result of this is shown in Figure 184 on page 223.

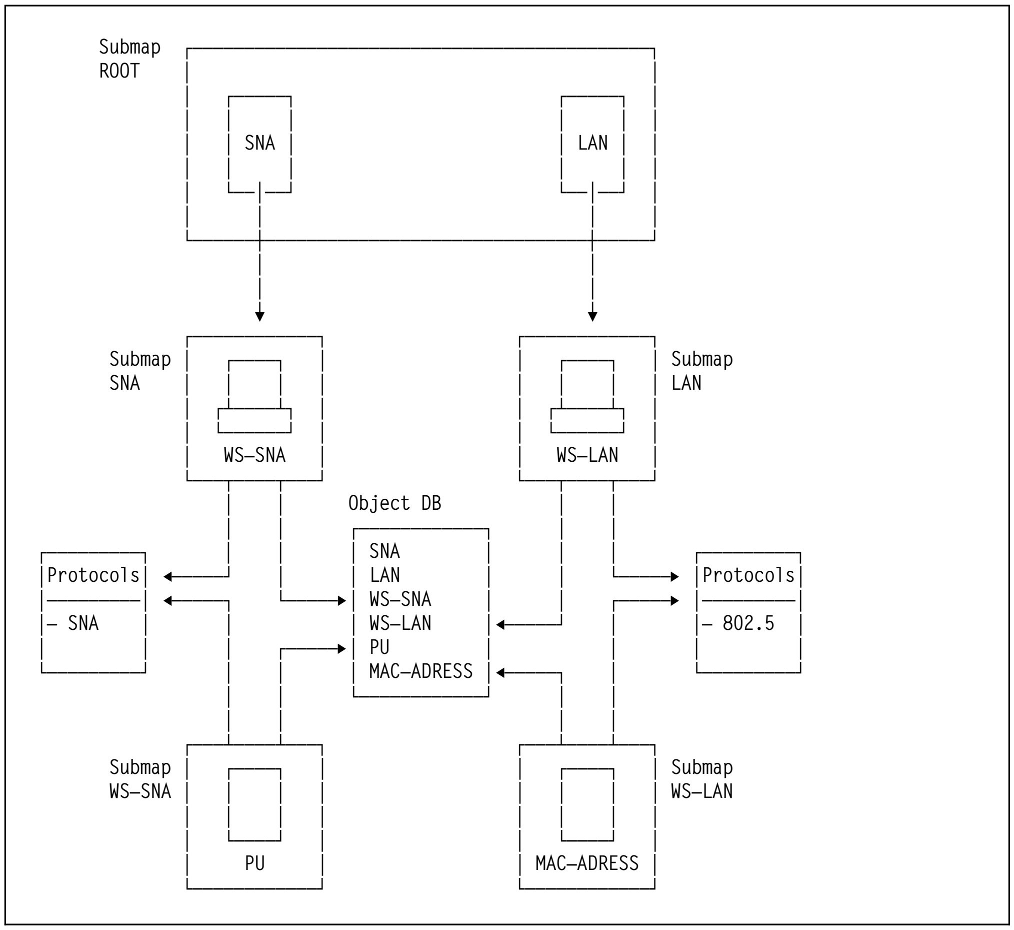


Figure 184. SNA and Physical Network Topologies with No Correlation

10.5.2 Open Topology Invocations

The previous scenario would require a series of calls to be made to the Open Topology interface (either as traps or API calls). The result of these calls is that objects are added to the topology and object databases.

For the SNA side of the diagram, the following sequence of calls would be made:

1. Trap newGraph for SNA with ISroot=True

   This causes gtmd to create an SNA object in the object database. xxmap adds an SNA submap symbol to the root map, associated with the SNA object in the object database. xxmap also creates an SNA submap in the map database.

2. Trap newGraph for WS-SNA with isRoot=False

   This creates an object for WS_SNA in the object database. xxmap cannot yet create a symbol for WS_SNA, since we have not yet told it which graph the WS_SNA graph is a member of. Therefore xxmap does not know the parentage of the symbol. It can, however, create a WS_SNA submap in the map database, associated with the WS_SNA object.

3. Trap newMember for WS_SNA

   xxmap can now create a symbol for WS-SNA in the map database, and place it on the SNA submap.

4. Trap newVertex for PU

   gtmd creates PU in the object database

5. Trap newMember

   This is to place a PU in the WS-SNA submap. gtmd creates a PU symbol associated with the PU object, and then places it into the WS-SNA submap.

10.5.3 Using Open Topology Correlation

Now that we have placed the SNA PU onto the submap, we want to correlate this PU with the MAC object that has been created by gtmd during the LNM/6000 discovery process. This is not done by LNM/6000 and SNA/6000, as the information about the existence of the other environment is not available to either of them. By contrast it *is* done by LNM/6000 and NetView for AIX IP support, because the MAC address of each IP interface is known, and can thus be correlated with its LNM/6000 equivalent.

If we know the mapping between the PU and its MAC address, we can provide correlation using the Open Topology MIB. To do the correlation, we need to send gtmd a newSAP trap, to indicate that the PU vertex object uses services provided by the MAC-ADDRESS vertex object. When this is done, the xxmap EUI application will take the following actions:

1. The WS_SNA object is linked to the WS_LAN object.
2. The PU symbol is copied into the WS-LAN submap.
3. The WS-SNA object is deleted.
4. The WS-SNA submap is deleted.

The effect on the NetView for AIX submaps and object data is shown in Figure 185 on page 225.

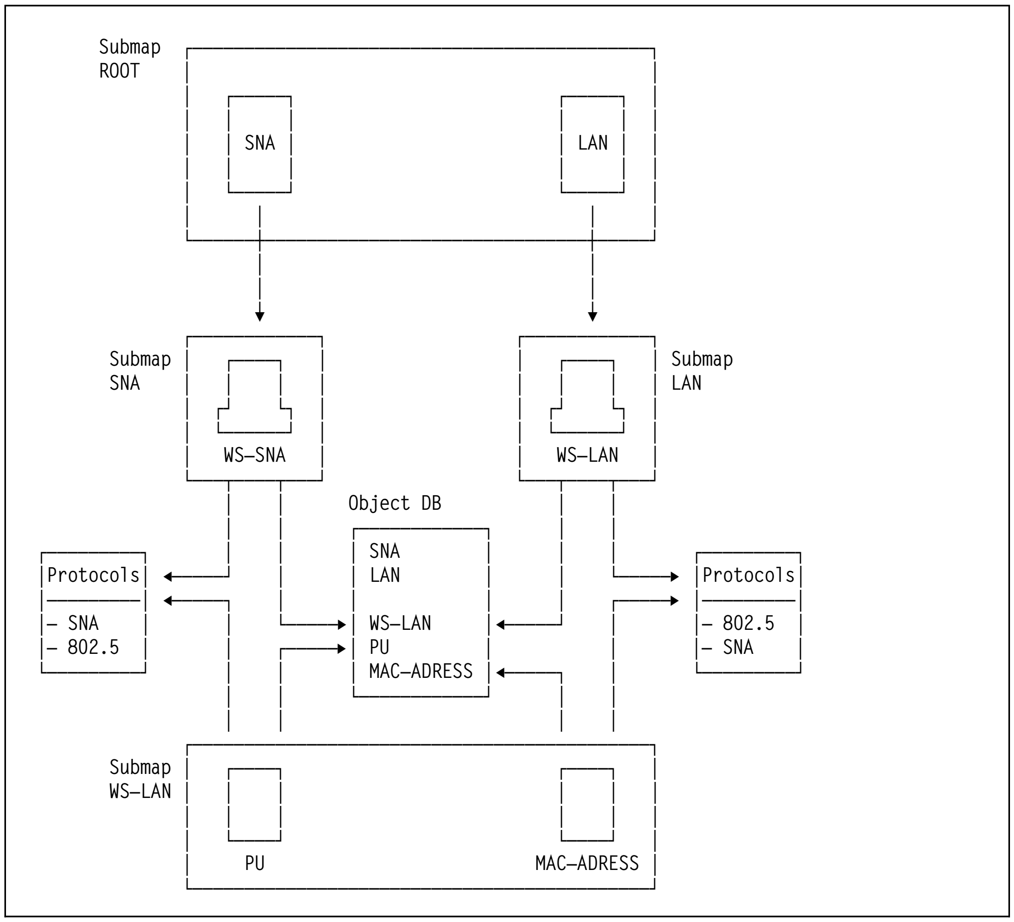


Figure 185. LNM/6000 and SNA/6000 with Correlation

The xxmap EUI application allows you to transfer between the SNA and the LAN submaps. This can be done by listing what protocols are being used by the workstation and switching to the one that you want.

10.6 The Open Topology API

Prior to NetView for AIX Version 3 we were only able to manipulate (create, delete, set attributes of) Open Topology objects using SNMP TRAPs. There are two main problems with this:

1. It is inefficient, in that several processing steps are needed to encode the trap, prepare and send it in a UDP frame, receive it, decode it, and finally take the action requested.
2. It is inherently unreliable, as it uses UDP. This could give us a problem when the traps arrive in the wrong order.

NetView for AIX now provides an API for Open Topology, which has a TCP socket connection to gtmd, thus overcoming these problems.

The API also improves the usability of the interface for the programmer. Using traps, the programmer has to keep track of index numbers associated with each object created. With the API, there is no need to worry about the index of the object that you create, as this is managed for you.

Finally, the API provides list functions, which allows the application to retrieve information about objects from the gtmd database.

10.6.1 Elements of the Open Topology API

There are three types of function provided by the Open Topology API:

- General-purpose routines. These start and stop the TCP socket connection between the calling code and gtmd, and control the attributes of the session.
- Error-processing routines. These report and interpret error codes from the API calls.
- Convenience routines. By far the largest group, these provide functions to add, delete, and set variables for any of the open topology objects previously described. There are also a number of get functions, which provide information about objects. The names of these routines are all self-descriptive. For example, nvotGetArcsInGraph returns a list of the arcs that are members of a given graph.

The API routines are fully described in *NetView for AIX Programmers Guide,* SC31-6238.

10.7 Open Topology Samples

In a previous project (documented in *Examples Using NetView for AIX*, GG24-4327), we created the wtotapi1 sample program that drives the NetView for AIX Open Topology processes. In this project we enhanced the wtotapi1 sample program.

You will find a listing of the sample program in Appendix D, "C Code for the wtotapi1 Sample Program" on page 271.

The objective of this program is to provide a simple command-line technique to define and control network topology views, using the Open Topology support of NetView for AIX. The program does not exploit the full capability of the API, but it does serve to show the sort of thing that may be achieved with little programming effort. To illustrate the use of the samples, we will use a worked example.

10.7.1 Worked Example Using Open Topology Sample Code

For our example we chose to represent the NFS distributed file system of several of the RISC System/6000s in the ITSO-Raleigh center.

Two machines act as NFS servers, providing disk access to three other RISC System/6000s.. Therfore, the elements we want to map on the NetView for AIX display are:

- The NFS servers
- The directories they export
- The NFS clients that mount the directories
- The file systems over which the directories are mounted

Ideally, we would issue commands to determine the relationships between these resources and then issue Open Topology requests to automatically generate the configuration submaps. However, for the purposes of our example we will simply define the topology statically.

10.7.1.1 Defining the Protocol ID

The first thing to consider is the protocol ID that we will use. As previously described in 10.3, “Terms and Concepts” on page 218, the protocol IDs are defined in the following two places:

- As an object ID in file /usr/OV/conf/oid_to_protocol
- As an integer value in the Open Topology MIB

We edited the oid_to_protocol file, adding the following entry:

```
″ITSO″=1.3.6.1.4.1.2.8.1
${ITSO}.1=″NFS″
${ITSO}.56=″IP″
```

The object ID (1.3.6.1.4.1.2.8.1) that we have assigned to ITSO is an experimental leg under the IBM enterprise ID in the MIB. In fact, the object ID used for the Open Topology protocol ID is not correlated with other MIB definitions, so it is not essential for it to be unique. However, it is good practice to use official OIDs where possible. If you are adding your own protocols in this way, you may wish to apply for an enterprise ID from the *Internet Assigned Numbers Authority*. IDs can be obtained via email from iana@isi.edu.

There is presently no way to add a vertex protocol number to Open Topology, so we will use 1 which has a description of Other.

10.7.1.2 Adding Symbols

We will want to use symbols that are not defined in the default /usr/OV/conf/C/oid_to_sym file, so before we start building the topology we need to update it. We added the following lines to oid_to_sym:

```
1.3.6.1.4.1.2.8.1.1:Network:Star
1.3.6.1.4.1.2.8.1.2:Computer:Workstation
1.3.6.1.4.1.2.8.1.3:Software:Process Tree
1.3.6.1.4.1.2.8.1.4:Software:Process
1.3.6.1.4.1.2.8.1.5:Device:Hard Disk
```

These lines associate a MIB object ID (in this case under the ITSO experimental leg) with symbols defined in NetView for AIX′s symbols directory: /usr/OV/symbols/C. The first name after the MIB OID is the *Symbol Class*; it is the name of a file in the symbols directory. The second name is the *Symbol Type*, which references an entry in the Symbol Class file. This in turn references a set of bitmaps.

10.7.1.3 Registering gtmd

Before you can work with the Open Topology you will have to register the gtmd daemon as follows:

- As root user change directory into /usr/OV/lrf.
- Still as root run the command `ovaddobj gtmd.lrf`.

You should receive the following message:

```
ovaddobj - Static Registration Utility
Successful Completion
```

10.7.1.4 Defining the Topology

Although the sample program wtotapi1 can be invoked from the command line, it is easier to place the commands in a file to be invoked together.

The command file used to create our NFS sample is shown in Figure 186.

```
prefix 1.3.6.1.4.1.2.8.1.
prot 1
add graph NFS Network:Star rowcol
focus NFS
add graph rs60003.nfs Software:Process tree
add graph rs60005.nfs Software:Process tree
focus rs60003.nfs
add box /usr/local Device:Hard Disk rowcol
add box /wtprint   Device:Hard Disk rowcol
add box rs60004    Computer:Workstation rowcol
add box rs60002    Computer:Workstation rowcol
add box rs60001    Computer:Workstation rowcol
add arc   rs60001 /usr/local
add arc   rs60002 /usr/local
add arc   rs60004 /usr/local
add arc   rs60002 /wtprint
focus rs60005.nfs
add box /usr/sys/inst.images Device:Hard Disk rowcol
add box /u/harald Device:Hard Disk rowcol
add box rs60004   Computer:Workstation rowcol
add box rs60002   Computer:Workstation rowcol
add arc   rs60004   /u/harald
add arc   rs60004   /usr/sys/inst.images
add arc   rs60002   /usr/sys/inst.images
```

Figure 186. Command File nfsmap Using wtotapi1 Sample Code

wtotapi1 reads commands from standard input, so we invoke the above command file by entering `wtotapi1 -p  nfsmap` (The ″-p″ option prevents printing of the Command> prompt).

The following are some notes on the contents of the nfsmap command file:

1. We want to use our own protocol OID (see 10.7.1.1, "Defining the Protocol ID" on page 227) instead of the default one. Therefore we tell wtotapi1 the dotted-decimal root for it.
2. We will use protocol instance 1 (NFS).
3. First we add a symbol to the root map to anchor our new topology. It will be a network symbol and the submap below it will use the Row/Column layout.
4. Next we want to add objects one layer further down the network hierarchy, so we define the parent object for them (the NFS root-graph that we just defined). In terms of the Open Topology model, the objects that we next add will be members of the NFS graph.
5. We add objects to represent the two NFS servers. Note that these are *graph graphs*, not *box graphs*, because we will not add vertices directly under them.
6. We have now gone one layer further down, and we add objects representing the directories exported by the NFS server on rs60003 and the machines that

mount them. Note that these are box graphs, since we will want to add vertices below them.

7. Next we establish the relationships between the exported directories and the machines using them, by adding connections (in the Open topology model, *arcs*) to the submap.

8. Finally we repeat steps 6 and 7 for the resources connected to the NFS server on rs60005.

The result of executing the previous sequence of instructions is shown in Figure 187.

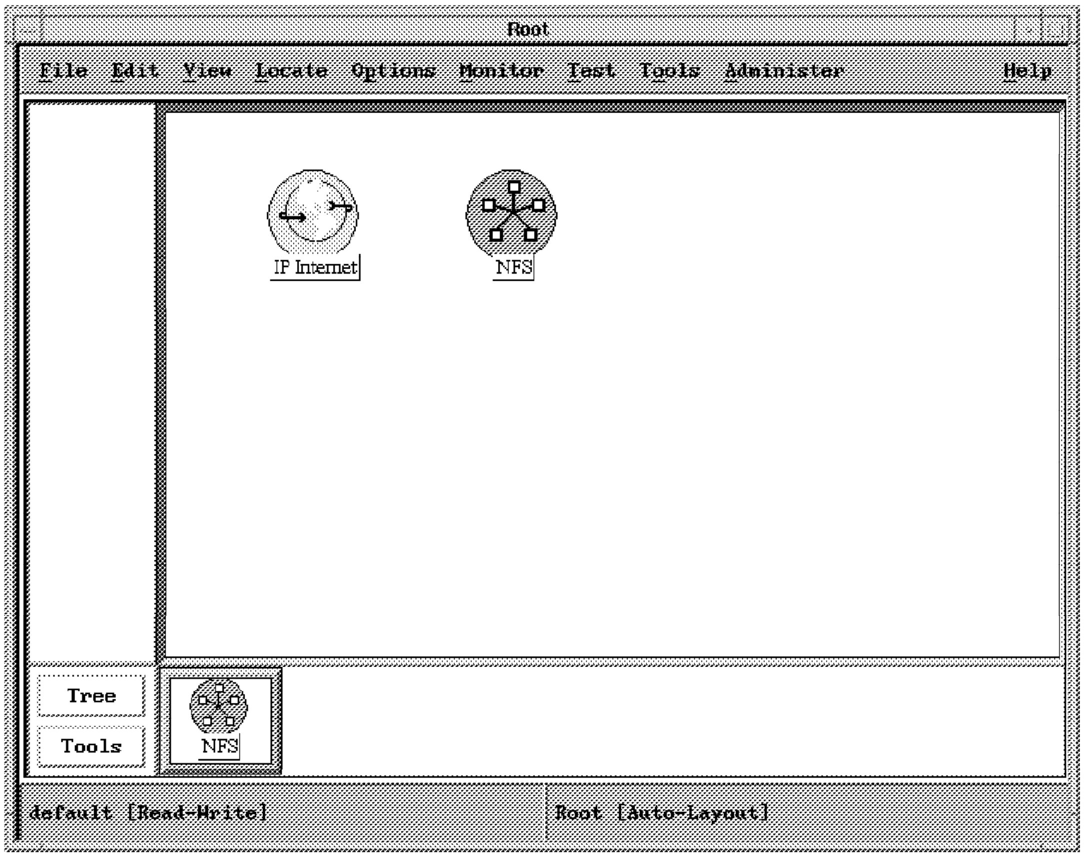


Figure 187. The Root Submap as Updated by this Example

The NFS symbol has been added by xxmap. There will also be an entry in the NetView for AIX object database representing this object.

Next we explode the NFS symbol with a double-click of the left mouse button. The panel shown in Figure 188 on page 230 is displayed.

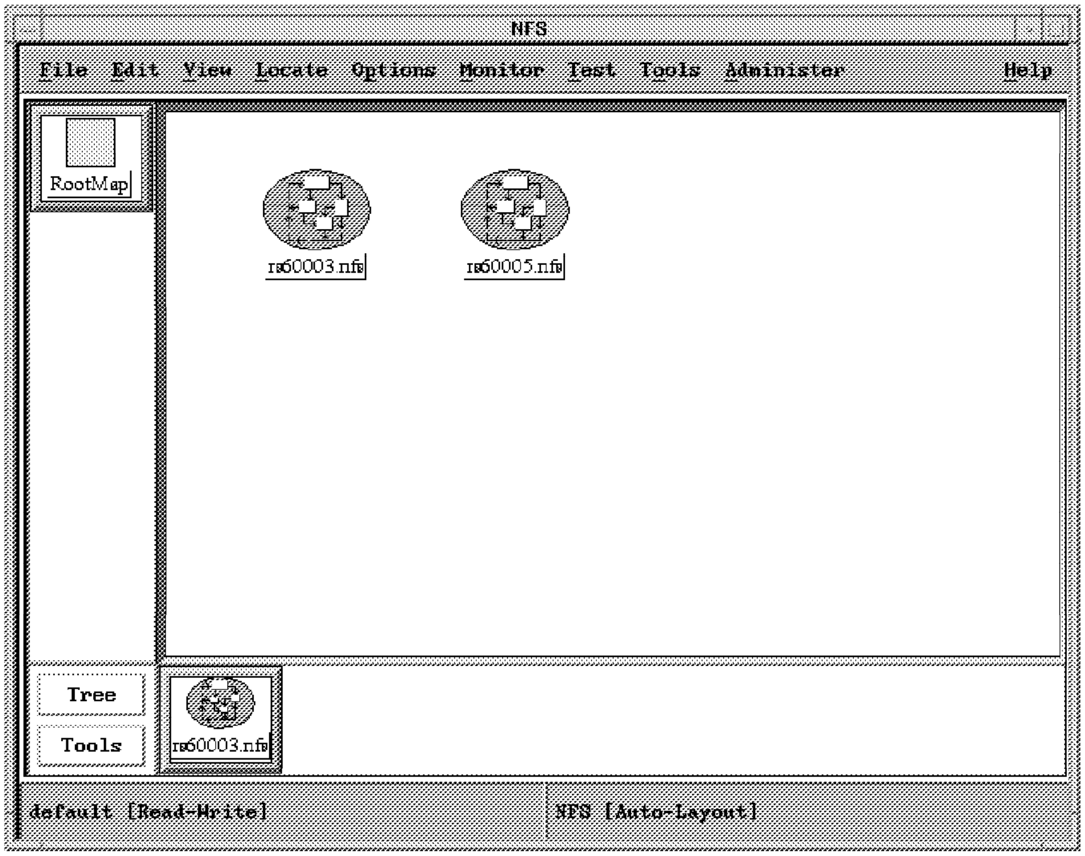


Figure 188. The NFS Server Submap

Our two servers are represented by Software:Process symbols. We then double-click on **rs60003.nfs** and the panel in Figure 189 on page 231 is displayed.

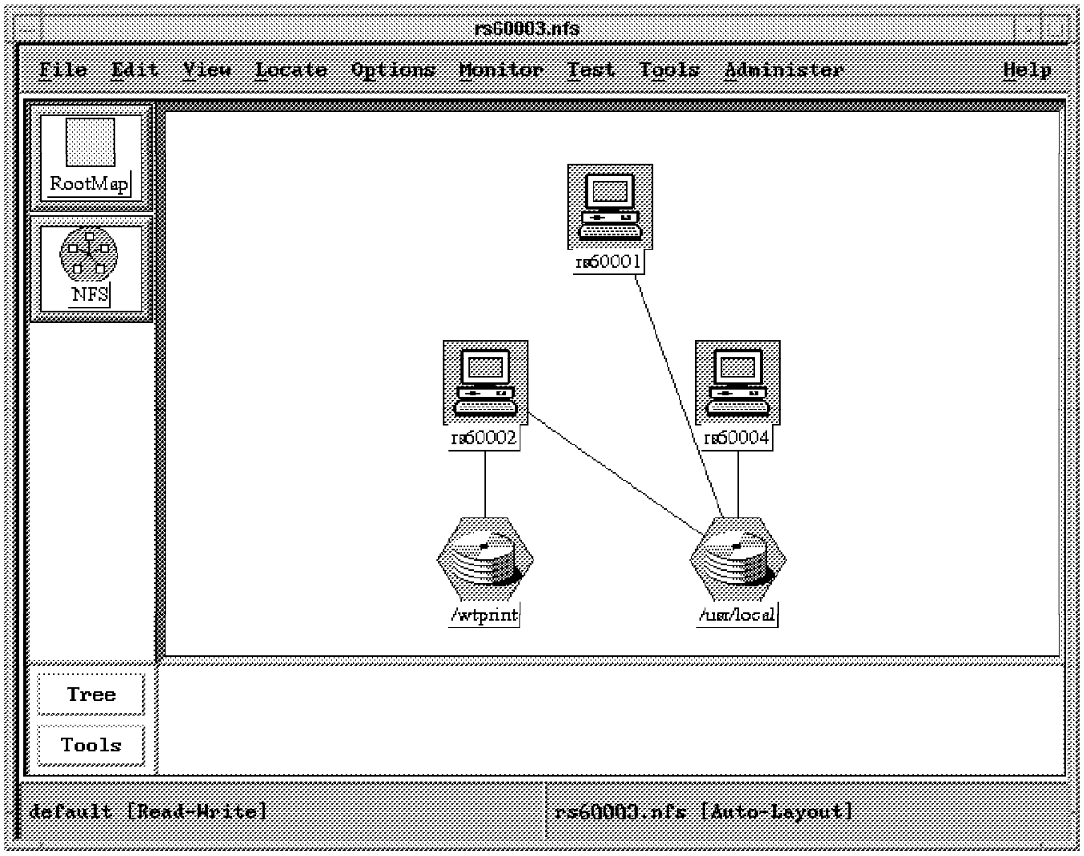

rs60003.nfs
File Edit View Locate Options Monitor Test Tools Administer Help
RootMap
NFS
rs60001
rs60002
rs60004
/wtprint
/usr/local
Tree
Tools
default [Read-Write]
rs60003.nfs [Auto-Layout]

Figure 189. The Mounted File System Connections

Here we see symbols representing the exported file systems and the NFS client systems that have mounted them.

10.7.1.5 Adding Status Representation

Thus far we have created a map of our NFS configuration, but all the symbols are blue (status unknown). We now want to add status representation to our NFS network views.

In the Open Topology model, status flows from the bottom upward. That is, only the elemental parts of the network (those lowest in the hierarchy) have status directly; all the higher-layer parts derive their status from the objects that are contained in them. The most basic network component in the Open Topology model is the vertex. If we want to add status representation to our network, therefore, we have to add vertices.

In the NFS network the lowest-level component is the file system (the real file system on the server and the mounted file systems on the clients). It is the status of these that we will monitor. To add vertices representing them, we pass the following commands shown in Figure 190 on page 232 to wtotapi1.

```
prefix 1.3.6.1.4.1.2.8.1
prot 1
focus /usr/local
add vertex rs60003:/usr/local Device:Hard Disk
focus /wtprint
add vertex rs60003:/wtprint Device:Hard Disk
focus rs60001
add vertex rs60001:/usr/local Device:Hard Disk
focus rs60002
add vertex rs60002:/usr/local Device:Hard Disk
focus rs60002
add vertex rs60002:/mnt/wtprint Device:Hard Disk
focus rs60004
add vertex rs60004:/usr/local Device:Hard Disk
focus /usr/sys/inst.images
add vertex rs60005:/usr/sys/inst.images Device:Hard Disk
focus /u/harald
add vertex rs60005:/u/harald Device:Hard Disk
focus rs60002
add vertex rs60002:/usr/sys/inst.images Device:Hard Disk
focus rs60004
add vertex rs60004:/usr/sys/inst.images Device:Hard Disk
focus rs60004
add vertex rs60004:/u/harald Device:Hard Disk
```

Figure 190. Commands to Add Vertices to NFS Submaps

Adding these vertices has the effect of populating the submaps below the box graphs in the lowest-layer submap (such as the submap shown in Figure 189 on page 231). For example if we explode the rs60002 symbol, we will see the screen in Figure 191 on page 233.

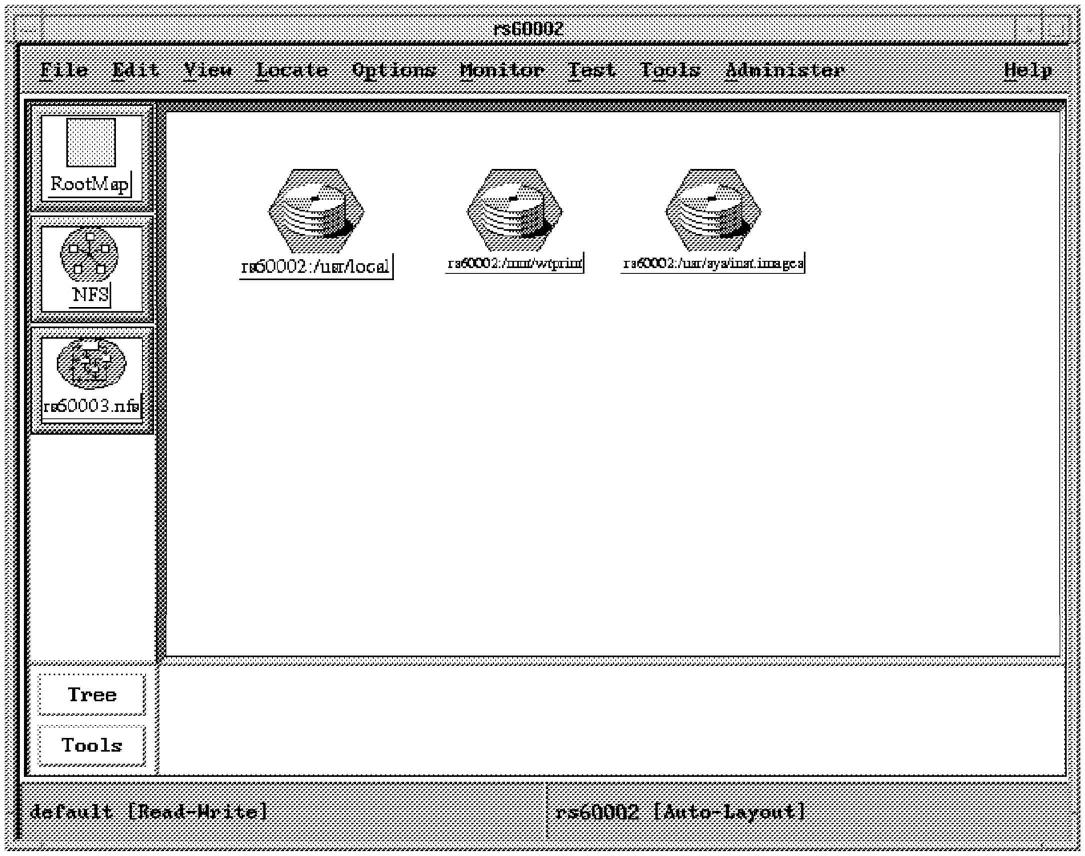


Figure 191. rs60002 Submap, Showing Vertex Symbols

When a vertex is created it gets a default status of normal (that is, up). This causes the symbol representing it to be green. This green status is propagated up the symbol hierarchy to the graph that contains it.

10.7.1.6 Setting Vertex Status

So far all of our work has been done on the manager (NetView for AIX). We have created a picture of the NFS network, but as yet it is only a static set of views. If our NFS application is to be useful, we need to have status information updated by monitors running on the agent systems.

The agent needs to have two components:

1. A method to detect the status change
2. A method to communicate this to the central manager

For the first component we could have used a simple shell script, to check for the existence of the file systems we are monitoring on a regular cycle. Instead, we chose to install Systems Monitor for AIX and configure the Threshold and File Systems MIB tables to perform this function. This monitoring capability is only one of the many functions of the Systems Monitor for AIX MLM agent. We will not explore the functions of the product here, but you may wish to read further about it in *AIX Systems Monitor Users Guide,* SC31-7042 and the redbook *IBM Systems Monitor Anatomy of a Smart Agent,* GG24-4398.

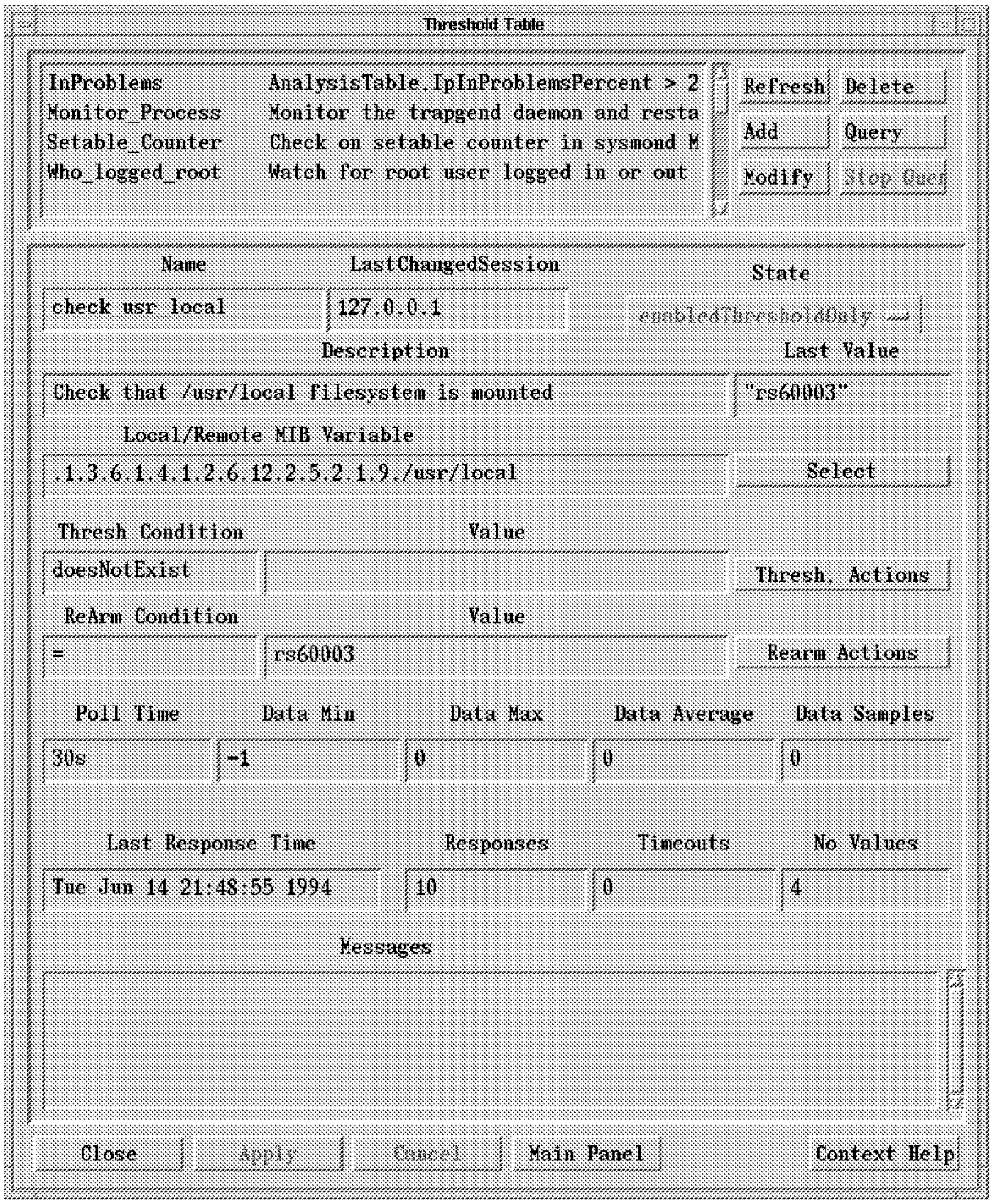
For the second component (communication of changes) we considered three possibilities, using standard functions of the Systems Monitor for AIX Threshold Table:

1. Send a trap and use event configuration in NetView for AIX to execute wtotapi1 when the trap arrives.
2. Place a copy of wtotapi1 on the agent system and use it to remotely update gtmd on the manager.
3. Execute the `snmptrap` command to set the status directly.

All options have the same effect (a vertex status change request is passed to gtmd on the NetView for AIX system). The differences lie in whether a TCP socket (wtotapi1) or SNMP trap (option 3) is the vehicle. We elected to go for option 3. The advantage is that because the snmptrap command is part of Systems Monitor, it will work on systems to which wtotapi1 may not be portable (for example, Sun Solaris or OS/2).

The Systems Monitor for AIX configuration screen for the Threshold Table is shown in Figure 192 on page 235. This entry causes the Systems Monitor agent on the target system (in this case rs60002) to poll every 30 seconds to see if the /usr/local file system is mounted, and to take action if it is not.

The threshold action is defined by clicking on the Threshold Actions button. Figure 193 on page 236 shows that we have told Systems Monitor for AIX to issue a command set_down. There is an equivalent action panel for when the threshold re-arms (which we have specified to be when the NFS mount is in place again).



Threshold Table

InProblems	AnalysisTable.IpInProblemsPercent > 2
Monitor Process	Monitor the trapgend daemon and resta
Setable_Counter	Check on setable counter in sysmond M
Who_logged_root	Watch for root user logged in or out

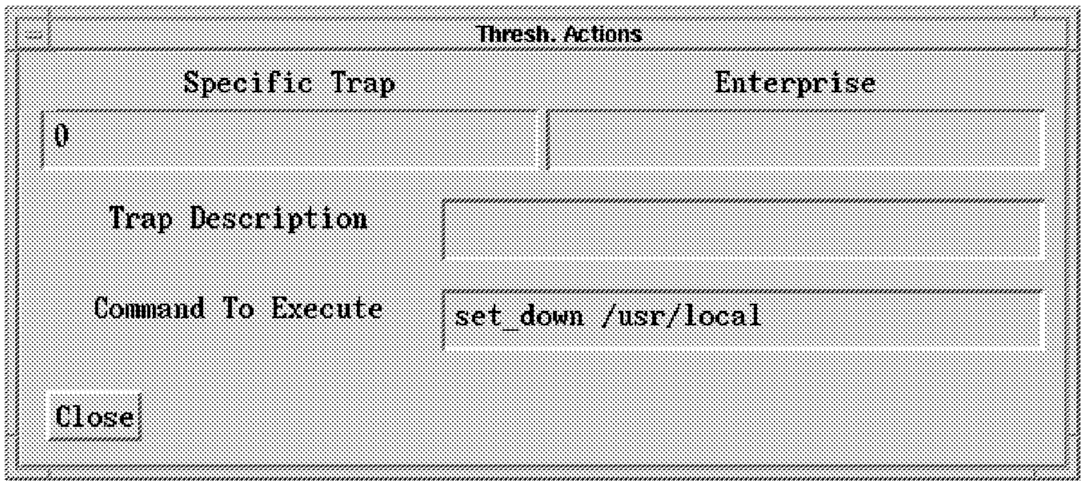
Name: check_usr_local
LastChangedSession: 127.0.0.1
State: enabledThresholdOnly
Description: Check that /usr/local filesystem is mounted
Last Value: "rs60003"
Local/Remote MIB Variable: .1.3.6.1.4.1.2.6.12.2.5.2.1.9./usr/local
Thresh Condition: doesNotExist
Value:
ReArm Condition: =
Value: rs60003

Poll Time	Data Min	Data Max	Data Average	Data Samples
30s	-1	0	0	0

Last Response Time	Responses	Timeouts	No Values
Tue Jun 14 21:48:55 1994	10	0	4

Messages

Figure 192. Systems Monitor Threshold Table Definition for NFS Monitoring



Thresh. Actions

Specific Trap: 0

Enterprise:

Trap Description:

Command To Execute: set_down /usr/local

Figure 193. Systems Monitor Threshold Action Definition. This action is triggered when the threshold defined in the previous figure is exceeded.

The set_down and set_up shell scripts each contain a single snmptrap command. The following is the set_down shell script:

```
us=`hostname`
resname=$us:$1
trap_tgt=rs60003
trapcmd=/usr/lpp/sm6000/original/snmptrap

$trapcmd $trap_tgt public .1.3.6.1.4.1.2.6.3.1 $us 6 1879048194 0 \
         .1.3.6.1.4.1.2.5.3.1.1.1.2 Integer 1 \
         .1.3.6.1.4.1.2.5.3.1.1.1.3 Octetstring $resname \
         .1.3.6.1.4.1.2.5.3.1.1.1.9 Objectidentifier 1.3.6.1.4.1.2.8.1.0  \
         .1.3.6.1.4.1.2.5.3.1.1.1.10 Integer 1 \
         .1.3.6.1.4.1.2.5.3.1.1.1.11 Integer 1 \
         .1.3.6.1.4.1.2.5.3.1.1.1.12 Integer 8 \
         .1.3.6.1.4.1.2.5.3.1.1.1.13 Integer 2
```

Figure 194. Shell Script set_down - Send Change Vertex Status Trap

The snmptrap command sends an enterprise specific trap, as defined in the Open Topology MIB. The specific trap number (1879048194) has the meaning *change vertex status*. The last four variables in the trap are the Operational Status, Unknown Status, Availability Status, and Alarm Status. Different combinations of these variables map to different NetView for AIX symbol statuses and, therefore, colors.

We enabled the Systems Monitor for AIX Threshold Table entry, and unmounted the /usr/local file system on rs60002. Within 30 seconds the color of the vertex symbol changed, and the change was propagated up the NFS submaps.

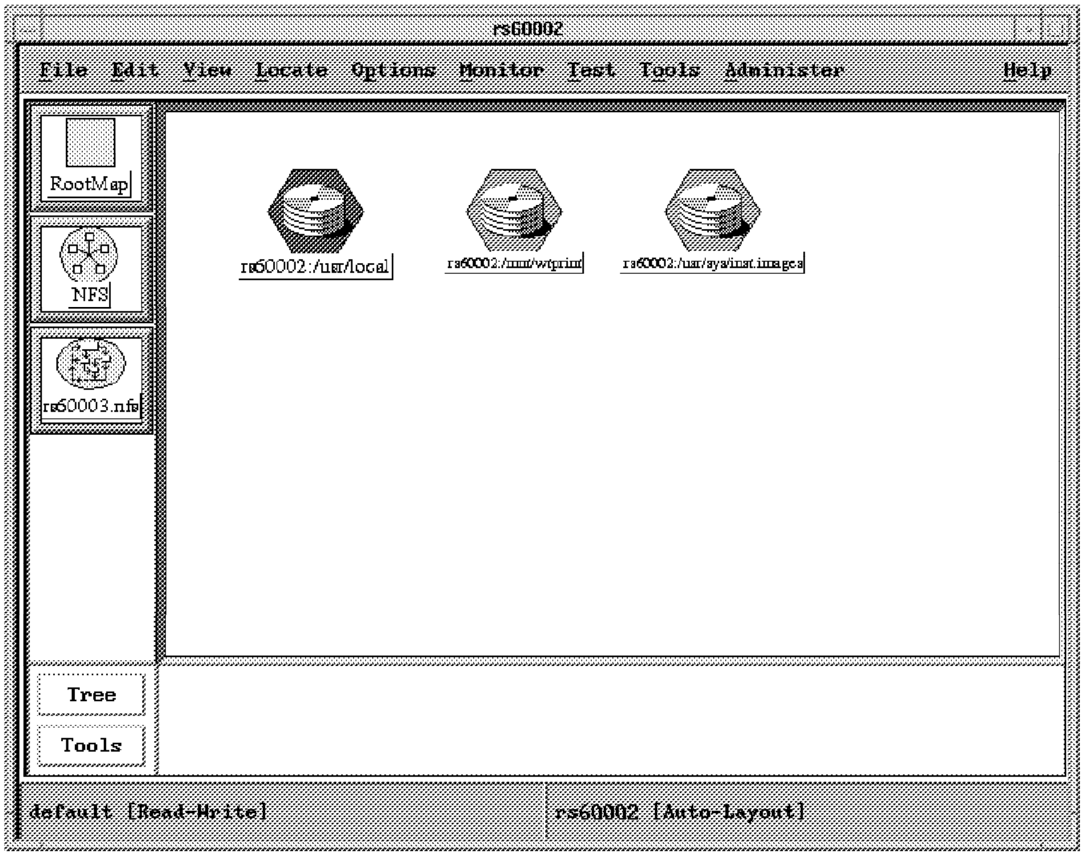


Figure 195. File Systems Submap with Status Change

10.7.1.7 Correlating with Other Protocols

The views of the NFS network that we have built are self-contained. The only place that they meet other protocols is on the Root submap. We now want to use the Open Topology Service Access Point function to provide correlation between NFS resources and IP network resources.

Although the IP network submaps are not generated by gtmd/xxmap, hooks have been placed in the Open Topology structure to enable correlation with IP resources to be made. The hooks take the form of one vertex for each IP interface whose name is the IP address of the interface and whose protocol number is 56. Note that it is the IP address that is used, even though the selection name by which the node is usually known may be an IP name from /etc/hosts or DNS.

We want our correlation to reflect the relationship between the node and the file systems in it (that is to say: *Node ″x″ contains NFS file system ″y″*). The way we achieve this is by generating SAP table entries, provided by NFS file system vertices and used by the node address. We used a file containing the following sequence of commands as input to wtotapi1:

```
prefix 1.3.6.1.4.1.2.8.1.
prot 1
add sap providing rs60001:/usr/local
add sap providing rs60002:/usr/local
add sap providing rs60004:/usr/local
prot 56
add sap using 9.24.104.26 rs60001:/usr/local 1
add sap using 9.24.104.28 rs60002:/usr/local 1
add sap using 9.24.104.27 rs60004:/usr/local 1
```

Figure 196. Adding SAP Entries Correlating NFS and IP. Note the two protocol IDs in use, 1 is our NFS protocol, 56 is the pre defined protocol ID for IP.

The most obvious result of this is that the submap below each node in the IP world and the NFS world are merged. For example, if we explode the rs60002 symbol from the IP Internet submap, we see a window similar to Figure 197 on page 238.

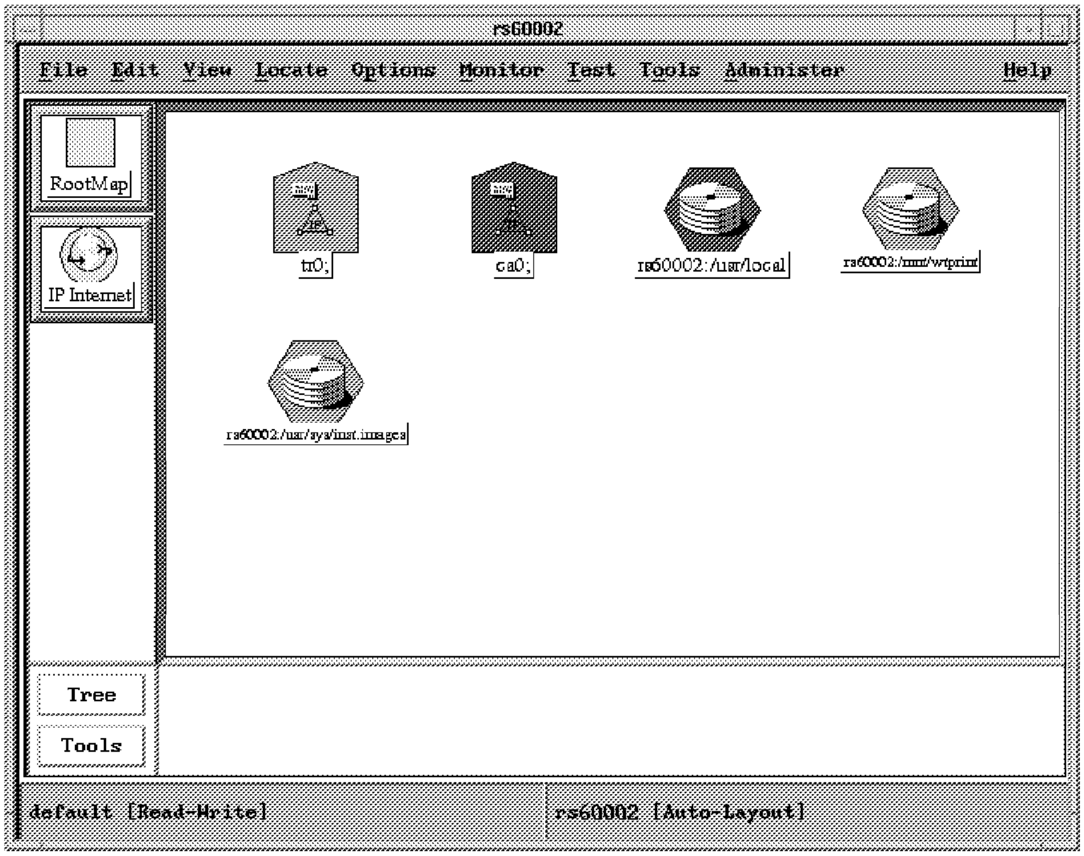


Figure 197. Merged Lowest-Layer Submap Due to SAP Correlation. Notice that both the NFS file systems and the IP interfaces can now be seen to be "part of" rs60002. Status changes to any of them may be propagated up both submap trees.

Another feature provided by SAP correlation is the protocol switching function. Under the View menu bar there is a Protocols. menu option, as shown in Figure 198 on page 239.

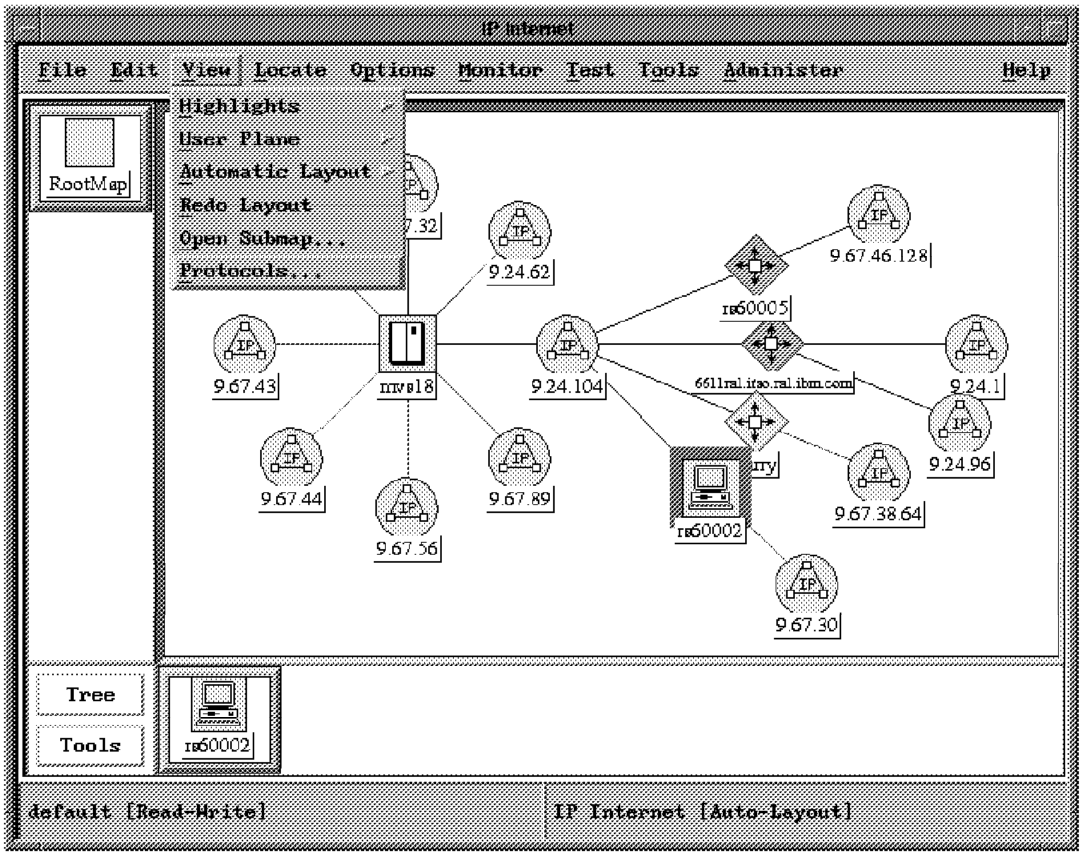
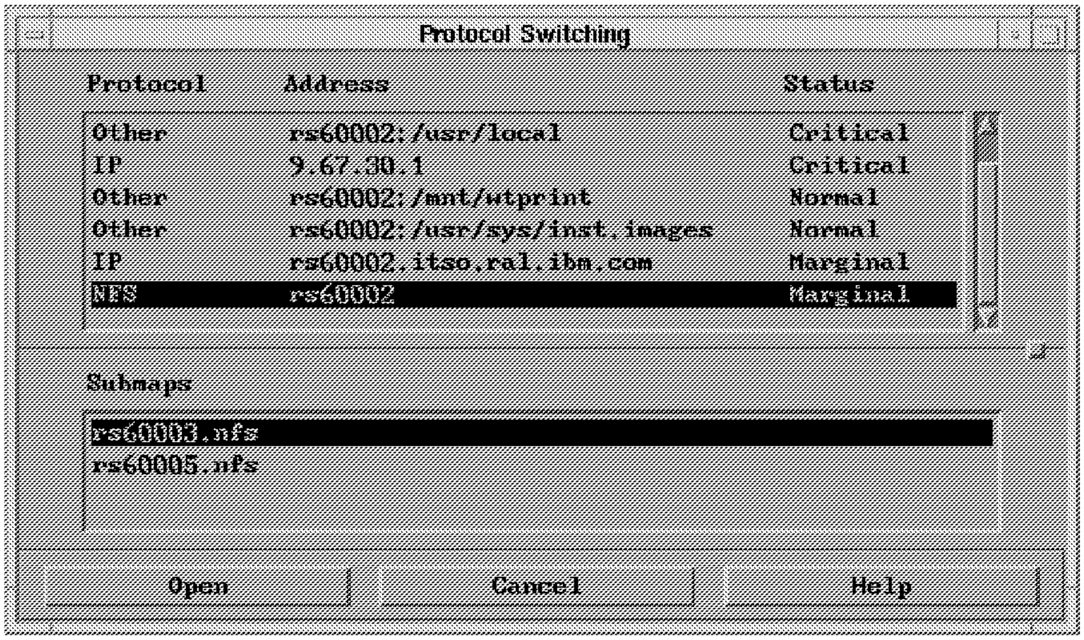


Figure 198. Protocol Switching Option

If we select rs60002 and select the **Protocols** option, we see the following panel (Figure 199 on page 239) from which we can directly pass to the different worlds in which rs60002 exists (in this case IP or NFS).


Protocol Switching

Protocol	Address	Status
Other	rs60002:/usr/local	Critical
IP	9.67.30.1	Critical
Other	rs60002:/mnt/wtprint	Normal
Other	rs60002:/usr/sys/inst.images	Normal
IP	rs60002.itso.ral.ibm.com	Marginal
NFS	rs60002	Marginal

Submaps

rs60003.nfs
rs60005.nfs

Figure 199. Protocol Switching Panel

Chapter 11. Report Generation for NetView for AIX

In *Examples of Selected Configuration and Customization Matters With NetView for AIX and Its Family*, GG24-2521, an example was shown of generating reports using NetView for AIX.

This chapter replaces the example from that document, using NetView for AIX Version 4 and adds a section on the use of snmpColDump.

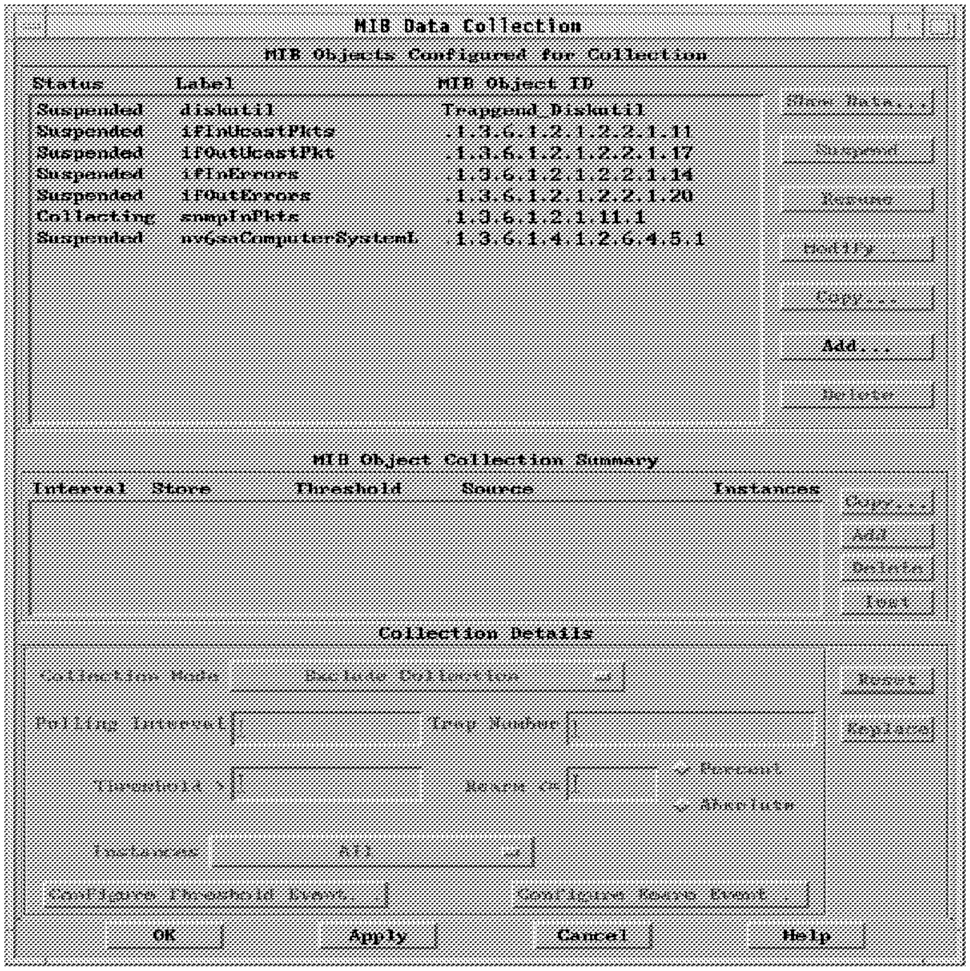
It is possible to collect data on any MIB value or MIB expression. In this example we will show a collection of the snmpOutPkts MIB.

11.1.1 Steps to Generate a Report For SNMPOutPackets

Before you can generate a report, you must collect the data that is used to generate the graph. NetView for AIX uses the snmpCollect daemon for data collection.

11.1.1.1 MIB Data Collection

Select **Tools --> Data Collection & Thresholds: SNMP** from the NetView for AIX menu bar and the window shown in Figure 200 on page 241 is displayed.



Status	Label	MIB Object ID
Suspended	diskutil	Trapgend_Diskutil
Suspended	ifInUcastPkts	.1.3.6.1.2.1.2.2.1.11
Suspended	ifOutUcastPkt	.1.3.6.1.2.1.2.2.1.17
Suspended	ifInErrors	.1.3.6.1.2.1.2.2.1.14
Suspended	ifOutErrors	.1.3.6.1.2.1.2.2.1.20
Collecting	snmpInPkts	.1.3.6.1.2.1.11.1
Suspended	nv6saComputerSystemL	.1.3.6.1.4.1.2.6.4.5.1

Collection Mode: Exclude Collection
Polling Interval:
Trap Number:
Threshold >:
Rearm <=:
Instances: All

Figure 200. MIB Data Collection Screen

Select **Add...** to add a new entry for snmpOutPkts.

Select **mgmt** and click on **Down Tree**.

Select **mib-2** and click on **Down Tree**.

Select **snmp** and click on **Down Tree**.

Select **snmpOutPkts**.

If you knew the numeric value for the MIB you could type this in directly. In this case the value is .1.3.6.1.2.1.11.2. At this stage you should change the label to a more meaningful value. We will use snmpOutPkts.

Select **OK** and you will get a new window as shown in Figure 201 on page 243.

The Source field shows the hosts for which the data is to be collected. It is possible to collect from specific hosts or on a group of hosts by using the collection facility. See Chapter 4, “The Collection Facility” on page 53 for details about using collections in NetView for AIX Version 4. In this case we will be collecting from all the nodes in two collections, mlmGroup and slmGroup.

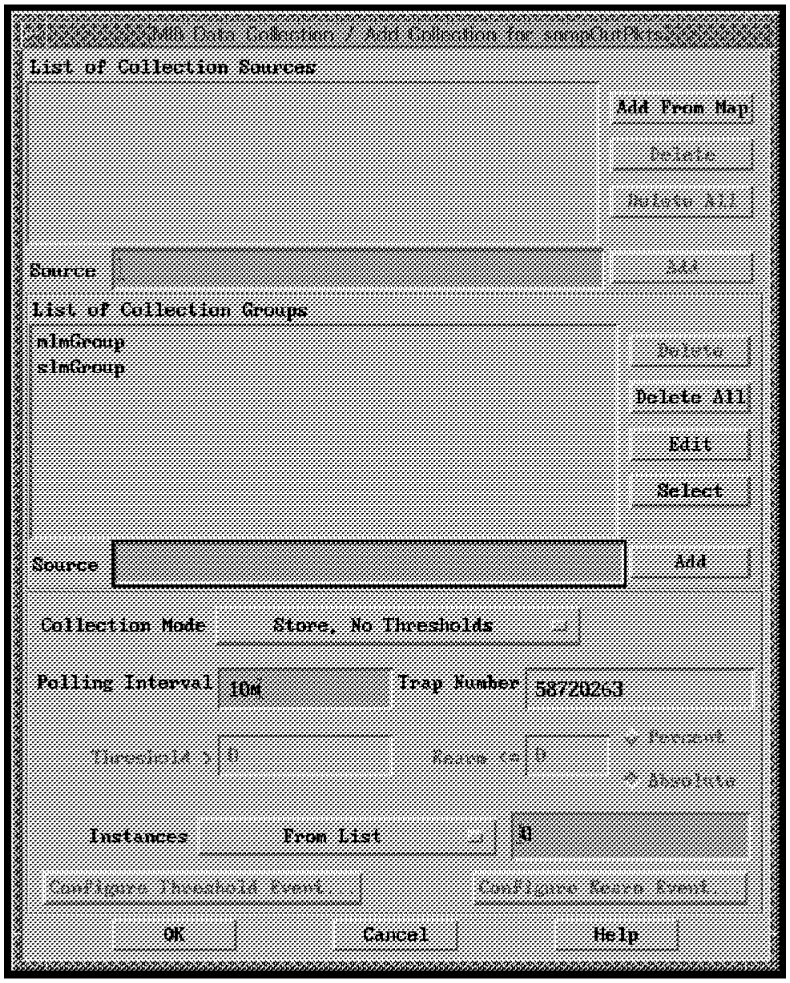
Click on **Add** to add your hosts to the list of collection sources or click on **Select** to add to the list of collection groups.

Click on **Collection Mode** to select the required value. In our example we want to store the data without checking it against a threshold.

Select the **Polling Interval** (in this case every 10 minutes).

Select the **Instances**. We selected to collect data from instance 0 only.

Click on **OK** when you are done.


MIB Data Collection / Add Collection for snmpOutPkts

List of Collection Sources

Source

List of Collection Groups

mlmGroup
slmGroup

Source

Collection Mode: Store, No Thresholds

Polling Interval: 10m
Trap Number: 58720263

Threshold >: 0
Rearm <=: 0
Percent
Absolute

Instances: From List
0

Figure 201. MIB Data Collection, Add Collection for snmpOutPkt

Now that the data is being collected it is possible to display it in two ways, as discussed in the following sections.

11.1.2 Generating reports from the NetView for AIX Version 4 EUI

The NetView for AIX Version 4 EUI allows you to graph the data by selecting **Tools ==> Graph Collected Data: SNMP** from the menu bar. From here it is possible to choose **Selected Nodes** or **All** This will add a graph of the selected data to the Control Desk. At any time it is possible to refresh the graph with the latest data by selecting **File ==> Update Data** from the menu bar.

To show the data in numerical form, return to the MIB Data Collection panel; select the entry that you are interested in and click on **Show Data**.

11.1.3 Generating the Report Script for a Graphical Report

It is also possible to create your own reports by building scripts. The scripts for the NetView for AIX reports are located in /usr/OV/reports/C. Figure 202 on page 244 shows the script we built for the snmpOutPkts report.

```
#!/bin/ksh
#
# EXAMPLE NAME: SNMPOutPackets
# FUNCTIONS:   Requires that the SNMP Data Collector first be run to
#              collect snmpOutPackets data values.  Use of the xnmgraph
#              facility to graph a specific MIB variable, snmpOutPackets.
#              No summarization is performed.  Only the nodes that are
#              currently selected will be graphed.  The graph will
#              represent a snapshot of the data collected.
# Explanation of the xnmgraph options --
# -browse will cause xnmgraph to look in /usr/OV/databases/snmpCollect
#         and not directly poll the target nodes.
# -title specifies the title of the window that xnmgraph runs in.
# -mib specifies each line on the graph.    The ′:′ separated options
#      expressed in -mib are (see the xnmgraph man page for details):
# 1. mibOID          .1.3.6.1.2.1.11.2
#       Dotted notation for snmpOutPkts.  This is the MIB variable whose
#       value we are graphing. This will cause the snmpOutPkts.<instance>
#       files, in /usr/OV/databases/snmpCollect, to be considered.
#       Actually the files can be named something other than snmpOutPkts.
#       snmpOutPkts is the default name.
# 2. mibLabel        SNMP Out Packets
#       Text string used to name the lines on the graph.  If not supplied
#       will default to the last textual component of mibOID(snmpOutPkts)
# 3. instanceRE      <default>
# 4. instMatchOID    <default>
# 5. instMatchRE     <default>
# 6. instLabelOID    <default>
# 7. instLabelTrunc  <default>
# 8. mult            <default>
#       Multiply all sample values by 1.  That is, no multiplication.
# 9. node            *
#       Report all nodes whose values have been captured
#       collections.
name=$(basename $0)
# Make sure that something has been selected
if [ -z ″$OVwSelections″ ] ; then
   ovxecho ″$name: Nothing has been selected from the map″
   exit
fi
# Make sure that there is not a : in the Selection name,
if [ -n ″`echo $OVwSelections | grep ′:′`″ ]; then
   ovxecho ″$name: Selected object $OVwSelections is not a node″
   exit
fi
exec xnmgraph \
  -browse \
  -title ″$name: Collected snmpOutPkts″ \
  -mib \
″.1.3.6.1.2.1.11.2::::::::\
$OVwSelections″
```

Figure 202. Script for Report snmpOutPkts of a Selected Node

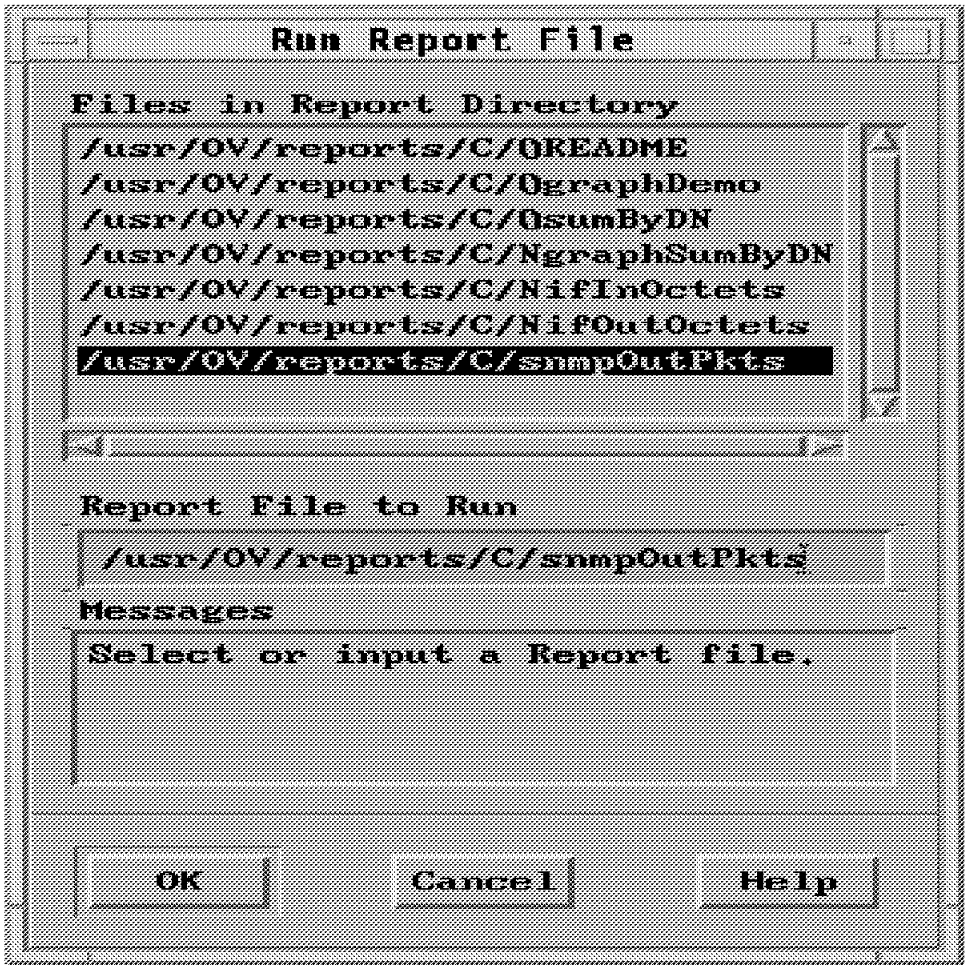
11.1.3.1 Activate a Report

The reports are activated from the NetView for AIX EUI.

Select the node for which you want the report.

Select **Monitor --> Reports: Site provided** from the menu bar to get a screen such as the one shown in Figure 203 on page 245.

Select **snmpOutPkts** from the list and click on **OK**.



Run Report File

Files in Report Directory

```
/usr/OV/reports/C/QREADME
/usr/OV/reports/C/QgraphDemo
/usr/OV/reports/C/QsumByDN
/usr/OV/reports/C/NgraphSumByDN
/usr/OV/reports/C/NifInOctets
/usr/OV/reports/C/NifOutOctets
/usr/OV/reports/C/snmpOutPkts
```

Report File to Run

/usr/OV/reports/C/snmpOutPkts

Messages

Select or input a Report file.

Figure 203. Run Report File Window

Now the xnmgraph command that is in the shell script is executed and displays a graph of the selected data. The graph will look like Figure 204 on page 246.

To print this graph, you can use the NetView for AIX Print Tool.

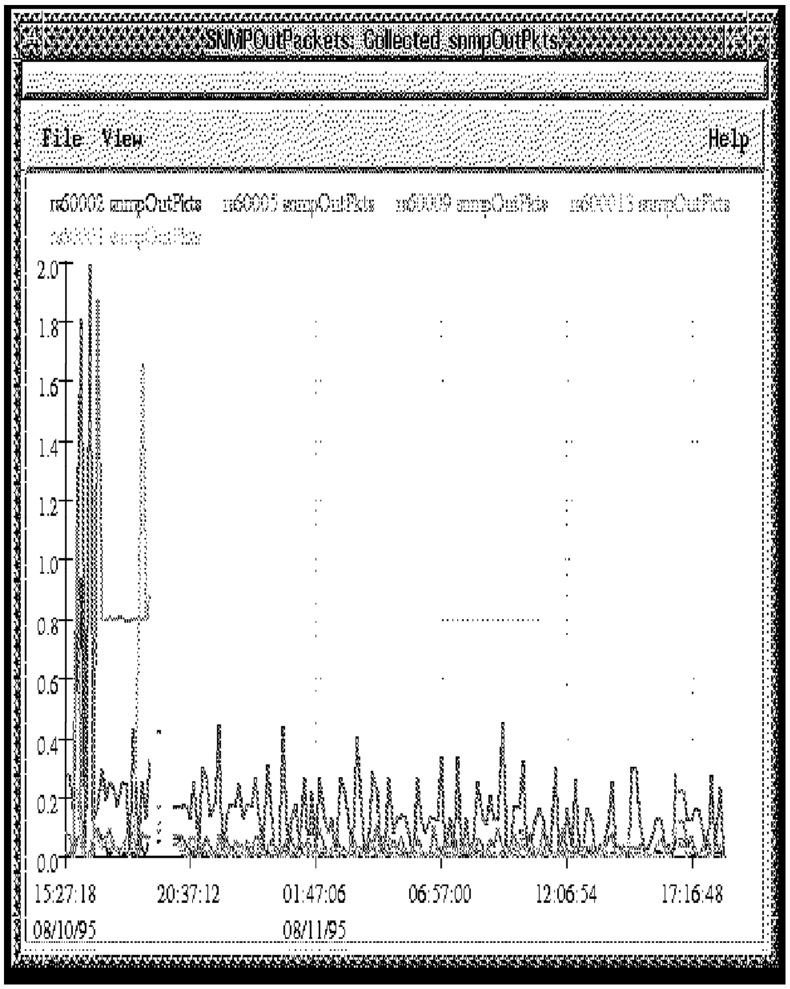


Figure 204. Result of the snmpOutPkt Report

11.1.3.2 Generating a Report Script for a List Report

To display the collected data as a graph for one specific node we invoked the xnmgraph function within a shell script, as described above. Another way to use the report function is to generate a report that is in a list or tabular format. In this example we show a list of the sum of SNMP out packets for all nodes from which we collected data. The list is sorted in descending order.

We are using the same data as for the previous report.

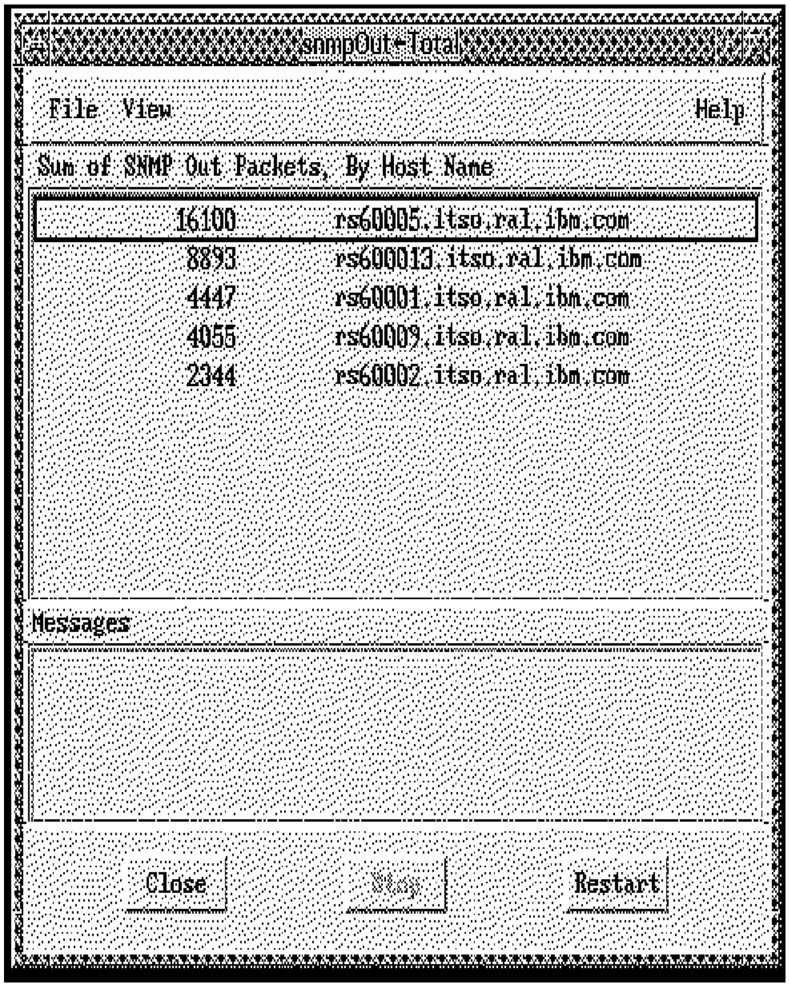
Figure 205 on page 247 shows the source of this report script.

```
#!/bin/ksh
# EXAMPLE NAME: snmpOutTotal
#
# FUNCTIONS: Requires that the SNMP Data Collector was run to collect
#            data values.  Sums the number of selected snmpOutPkt values,
#            by Node name, from Data Collector data.  Sorts in
#            descending order, by sum and displays on standard output.
#            This would provide you, at a gross level, which nodes have
#            the most SNMP traffic.
#            All nodes for which data has been collected will be reported
name=$(basename $0)
workfile=/tmp/$name$$
trap "rm -f $workfile; exit" 0 1 2 3 15 # Remove workfile on exit
cd /usr/OV/databases/snmpCollect      # Go to where the collections are
   snmpColDump -Tt snmpOutPkts.0 |
awk -F\t '{
# Input Data Fields are:
# $1  -  Start Time, in ASCII
# $2  -  Domain Name
# $3  -  Collected Data
# $4  -  Start time in seconds
# $5  -  Stop time in seconds
sum[$2]+=int($3*($5-$4))  # Convert the value collected, which is in
                          # changes per second and sum, indexed by Domain
                          # name.  Value * (StopTime - StartTime)
}
END {
   for( i in sum )              # Unload the sum array
      printf( "%16.0lf\t%s\n", sum[i], i )
}' | sort +0 -nr > $workfile    # Put into descending order, by sum

# Wrap the output in a resizeable Motif window for good looks
# (don't exec xnmappmon, or trap won't get invoked)
xnmappmon -commandTitle "$name" \
   -commandHeading "Sum of SNMP Out Packets, By Host Name" \
   -cmd cat $workfile
```

Figure 205. Script /usr/OV/reports/C/snmpOutTotal

When you invoke this new report via the EUI as previously described, you will get a list of all nodes with the sum of SNMP output packets for each one during the collected time interval. Figure 206 on page 248 shows an example of this list.

To print this list, click on **File --> Print...** in the window and enter your print command.



Sum of SNMP Out Packets, By Host Name

```
    16100   rs60005.itso.ral.ibm.com
     8893   rs600013.itso.ral.ibm.com
     4447   rs60001.itso.ral.ibm.com
     4055   rs60009.itso.ral.ibm.com
     2344   rs60002.itso.ral.ibm.com
```

Messages

Figure 206. Result of the snmpOutTotal Report

To generate this report on a monthly basis, simply remove the data collection input (in /usr/OV/databases/snmpCollect) after the report is generated, so you will display the data for only one month.

Note: You can also start each report from the command line or from another program (cron, for example), so you can easily automate the report function.

11.1.3.3 Generating the snmpColDump ASCII File

The data generated by Data Collection and Thresholds is stored in binary form. If you need to load this data into another program such as a spreadsheet or database it is necessary to convert it into a readable format. The NetView for AIX application provides the snmpColDump command to do this. The command to convert the snmpOutPkts example would be:

```
snmpColDump /usr/OV/databases/snmpCollect/snmpOutPkts.0 > /tmp/snmpOutPkts.ascii
```

This will produce a text file containing the data in ASCII format. Figure 207 on page 249 shows a partial output of this command.

```
08/10/95 15:37:18  rs60002.itso.ral.ibm.com  0.00166658
08/10/95 15:37:18  rs60005.itso.ral.ibm.com  0.27332
08/10/95 15:37:18  rs60009.itso.ral.ibm.com  0.0766641
08/10/95 15:37:18  rs600013.itso.ral.ibm.com 0.0133333
08/10/95 15:47:18  rs60002.itso.ral.ibm.com  0.0316667
08/10/95 15:47:18  rs60005.itso.ral.ibm.com  0.161667
08/10/95 15:47:18  rs600013.itso.ral.ibm.com 0.020001
08/10/95 15:47:18  rs60009.itso.ral.ibm.com  0.00166669
08/10/95 15:57:18  rs60002.itso.ral.ibm.com  0.82
08/10/95 15:57:18  rs60005.itso.ral.ibm.com  1.06498
08/10/95 15:57:18  rs600013.itso.ral.ibm.com 1.11004
08/10/95 15:57:18  rs60009.itso.ral.ibm.com  0.0633344
08/10/95 15:57:24  rs60001.itso.ral.ibm.com  0.226667
08/10/95 16:07:18  rs60002.itso.ral.ibm.com  0.938333
08/10/95 16:07:18  rs60005.itso.ral.ibm.com  1.8033
08/10/95 16:07:18  rs600013.itso.ral.ibm.com 0.176676
08/10/95 16:07:18  rs60009.itso.ral.ibm.com  0.0666667
08/10/95 16:07:24  rs60001.itso.ral.ibm.com  0.0283333
08/10/95 16:17:18  rs60002.itso.ral.ibm.com  0.00166348
08/10/95 16:17:18  rs60005.itso.ral.ibm.com  0.17
08/10/95 16:17:18  rs600013.itso.ral.ibm.com 0.00166672
08/10/95 16:17:18  rs60009.itso.ral.ibm.com  0.00166669
08/10/95 16:17:24  rs60001.itso.ral.ibm.com  0.00166667
```

Figure 207. Partial Ascii Output of Default snmpColDump

The first two columns are the date and time the collection was made. The third column is the resolved name of the host from which the collection was taken and the last column is the value of the data. Further information on the snmpColDump command can be found in the man entry. A copy is shown in Figure 208 on page 250.

Now that the file is in ASCII format, it is necessary to transfer it to your system. This is beyond the scope of this book but some possible methods would be to use NFS, FTP or NetView Distribution Manager.

If your NetView for AIX Version 4 is configured to use a relational database, it is possible to convert the ASCII files and run queries using SQL. This technique is further explained in *Netview for AIX Database Guide*, SC31-8167. There are also some examples of using the SQL support in *Examples of Using NetView For AIX*, GG24-4327.

```
----------------------------------------------------------------
snmpColDump(1)
----------------------------------------------------------------


Purpose

Dumps, in ASCII, or modifies data collected by snmpCollect


Syntax

snmpColDump [-TIpfstoiu] [-l Object-ID] [-L Object-ID] filename
snmpColDump [-r asciifile filename]



Description

The snmpColDump command formats a single binary data file in
/usr/OV/databases/snmpCollect that was previously created by
snmpCollect into human-readable and machine-parseable format.  It
also supports editing of the binary data.

For each MIB variable being collected, snmpCollect creates the
following two files in /usr/OV/databases/snmpCollect:

o   A binary data file containing the data collected for that
    variable.  The binary file has the same name as the label
    specified in the MIB Data Collection menu.

o   A separate ID file containing the units, syntax, and object
    ID for the MIB variable.  This ID file has the same name as
    the binary data file, except there is an ! appended to the
    filename.  The information in the ID file is also available
    by way of various options to snmpColDump.
```
Figure 208 (Part 1 of 6). snmpColDump man Page (August 1995)

```
The default output from snmpColDump consists of records, one per
line, containing the following three tab-separated fields:

o   Date and time when the data was collected

o   Domain name of the node on which the data was collected

o   Value of the collected variable.  MIB variables of type
    COUNTER are printed in floating-point notation.  COUNTER
    values are recorded as changes-per-second.  INTEGER and GAUGE
    types are printed as whole (non floating-point) numbers.


Flags

The following options are used to add additional fields to
snmpColDump output.  To remove an unwanted field, use awk(1).

-t
    Displays the time the data was collected as seconds since
    1970.  This makes sorting records easier.

-T
    Displays the start time as seconds since 1970.

-I
    Displays the official IP address of the node.

-p
    Displays the collection period in seconds.

-l Object-ID
    Displays an instanceString which is obtained through SNMP.
    SNMP getnext requests are made to the current node using this
    object ID and the instance of the Object-ID whose value
    matches the instance of the collection is displayed.  See the
    following examples for use.

    Note:  When you use the -l option, if the SNMP request cannot
           get the requested data, it will create the string
           (Instance_n).
```

Figure 208 (Part 2 of 6). snmpColDump man Page (August 1995)

```
-L Object-ID
    Displays a value which is obtained through SNMP.  The
    instance is appended to the Object ID given as the argument
    to this option.  An SNMP get request is made to the current
    node using this object ID and the resulting value is
    displayed.  Note that embedded blanks and tabs may be in this
    returned value, which is why it is the last field displayed.
    See the examples for information about usage.

    Note:  When you use the -L option, if the SNMP request cannot
           get the requested data, it will create the string
           instance_n.

filename
    Specifies path to the binary data file that is given as an
    argument to snmpColDump.

-r asciifile
    Replaces the binary file filename with the binary equivalent
    of asciifile.  If asciifile is -, standard input is used.
    This allows truncation of files, or modification of the
    contents of the file.  The asciifile must be an ASCII file in
    the form of lines containing

           <startTime> <endTime> <Dotted IPaddress> <value>

    where <startTime> and <endTime> are integers representing
    time since the epoc (time_t format), <Dotted IPaddress> is
    the string containing the IP address, and <value> is the
    double precision value for the interval.  An example of this
    form can be obtained using the command

      snmpColDump -tTI FILE |
      awk -F\t '{printf("%d\t%%s\t%lg\n", $4, $5, $6, $3)}'

    This allows deleting of old entries in the binary data base
    file.
```

Figure 208 (Part 3 of 6). snmpColDump man Page (August 1995)

```
The following options to snmpColDump help you gather additional,
specific information for generating reports, but produce no data:

-s
    Displays type (syntax) string of the MIB variable.

-o
    Displays objectID string of the MIB variable.

-i
    Displays instance of the MIB variable.

-u
    Displays units string of the MIB variable.

The following option to snmpColDump enables you to continually
monitor the data collected by snmpCollect:

-f
    Follows the file, sleeping until more data is available to
    display.


Implementation Specifics

The environment variable LANG determines the language in which
messages are displayed.  If LANG is not specified, or is set to
the empty string, the default C is used instead of LANG.  If any
internationalization variable contains a setting that is not
valid, snmpColDump behaves as if all internationalization
variables are set to C.

The snmpColDump command supports single-byte character code sets.
```

Figure 208 (Part 4 of 6). snmpColDump man Page (August 1995)

```
Examples

The following examples assume that data have been collected using
snmpCollect under the label ifInOctets.1.  It also assumes use of
sh(1) or ksh(1).

If you want to print out a the average value for node Moe, enter:



      snmpColDump /usr/OV/databases/snmpCollect/ifInOctets.1 |
           awk -F\t '/Moe/{num++; sum+=$3} END{print
  sum/num}'

If you want to retrieve information about node Curly, enter:

      snmpColDump /usr/OV/databases/snmpCollect/ifInOctets.1 |
           fgrep Curly

If you want to additionally display the description of the
collected interface, enter:

      snmpColDump -L interfaces.ifTable.ifEntry.ifDescr \
           /usr/OV/databases/snmpCollect/ifInOctets.1

If you want to additionally display the IP address, of the
interface, which is obtained as an instanceString from the
ipAdEntIfIndex instance whose value matches the current instance
of ifOctets, enter:

      snmpColDump -l ip.ipAddrTable. ipAddrEntry.ipAdEntIfIndex \
           /usr/OV/databases/snmpCollect/ifInOctets.1
```

Figure 208 (Part 5 of 6). snmpColDump man Page (August 1995)

If you want to keep only the last 2000 entries in file 1MinLoadAvg, enter:

```
lineno=′snmpColDump 1MinLoadAvg | wc -l′
if [ $lineno -gt 2000 ]; then
   lineno=′expr $lineno - 2001′
else
   lineno=1
fi
# In the awk program:
# $4=startTime, $5=endTime, $6=dottedIPAddr, $3=value
snmpColDump -tTI 1MinLoadAvg | sed -n $lineno′,$p′ |
awk -F\t ′{printf(″%d\t%d\t%s\t%lg\n″, $4, $5, $6, $3)}′ |
snmpColDump -r - 1MinLoadAvg
```

Files

/usr/OV/databases/snmpCollect/*

/usr/OV/databases/snmpCollect/ *[!]

Related Information

See ″snmpCollect(8).″

Figure 208 (Part 6 of 6). snmpColDump man Page (August 1995)

Appendix A. How to Obtain the Samples in this Book

The C code, shell scripts and event ruleset examples in this book are all available using anonymous FTP, as follows:

For Users Within the IBM TCP/IP Network

- Connect to rsserver.itso.ral.ibm.com via FTP.
- Specify a user ID of anonymous.
- Enter your internet mail address in place of a password.

You will find the samples in the file /pub/sg244515.tar.Z. This is a compressed tar archive containing all the files. Unpack it in a temporary directory. You will also find a file named sg244515.README in the same directory, which contains a decription of the package contents. This README file is shown in Figure 209 on page 258.

For Users Outside the IBM TCP/IP Network:

- Connect to ftp.almaden.ibm.com via FTP.
- Specify a user ID of anonymous.
- Enter your internet mail address in place of a password.

You will find the samples in file the /redbooks/SG244515/sg244515.tar.Z. This is a compressed tar archive containing all the files. Unpack it in a temporary directory. You will also find a file named sg244515.README in the same directory, which contains a decription of the package contents. This README file is shown in Figure 209 on page 258.

```
NetView for AIX V4 - Samples

This collection of stuff is from redbook SG24-4515, ″Examples using NetView for AIX
Version 4″.  The authors were Rob Macgregor (ITSO-Raleigh), Dave Shogren (ITSO-RALEIGH),
Randy Craig (IBM US, RTP), Luigi Casagrande (IBM Italy), Massimo Carnevali (IBM Italy),
Sergio Costa Lage (IBM Brazil), Blaz Mertelj (IBM Slovinia), Paul Fearn (IBM UK) and
Mark Hodge (IBM UK).

There are a lot of sample programs and event rulesets here to help you
make good use of the new features in NetView for AIX V4.

We don′t give any guarantees for the code, and there are almost certainly
better ways to do it - please send any comments to Rob Macgregor (MCGREGOR
at WTSCPOK or robmacg@vnet.ibm.com) or Dave Shogren (SHOGREN at WTSCPOK or
shogren@vnet.ibm.com).

In this collection you will find the following files:

rulesets         A subdirectory containing all the ruleset samples from the book

wtcoll.c         A utility program that exploits the collections API

wteuiap6         A set of programs that use the OVw API to allow you to create
                 network topologies from command line. This is the latest incarnation
                 of a package that has been published in several previous redbooks.
                 It has been modified to support client/server operation.

wtotapi1         A program that exploits the open topology API. This also was
                 published previously. This version fixes some problems with the
                 earlier one.

wtdepend         A package of programs, shell scripts and rulesets that allow you
                 to set up dependencies between routers and nodes ″behind″ them.
                 This means that you can suppress node down messages when the
                 router is down.

Several of the tar files and directories have further READMEs inside them.
```

Figure 209. Readme File for the Samples Package

Appendix B. C Code for the wtcoll Sample Program

```
/*---------------------------------------------------
   wtcoll.c - General purpose program to manipulate objects in a
              collection from the command line.

   From:     : Redbook "Examples Using NetView for AIX V4"
               (SG24-4515)

   Written by: Sergio Costa Lage (IBM Brazil)
             : Luigi Casagrande (IBM Italy)
          and: Rob Macgregor (IBM ITSO-Raleigh)

   For further information, contact:

   Rob Macgregor or Dave Shogren, ITSO-Raleigh.
   robmacg@vnet.ibm.com or shogren@vnet.ibm.com

   Copyright (C) IBM Corporation 1995
---------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <OV/ovw.h>

#include "/usr/OV/include/OV/ovw_obj.h"
#include "/usr/OV/include/OV/nvCollection.h"
#include "/usr/OV/include/OV/nvCollectionErrs.h"

#define ID              0
#define SELECTION_NAME  1

OVwMapInfo *map;

   /*
      Function Definitions
   */

void usage()
{
  fprintf( stdout, "\n========================================================================\n" );
  fprintf( stdout, "              >>>>>>>     wtcoll program     <<<<<<<\n\n" );
  fprintf( stdout, "Please choose one of the following options:\n\n" );
  fprintf( stdout, "(1)  [-createField] to create a database field in an object\n" );
  fprintf( stdout, "     parameters: selection name, collection name, field name, value\n");
  fprintf( stdout, "\n");
  fprintf( stdout, "(2)  [-setField]    to change a field value in all objects of a\n" );
  fprintf( stdout, "                    collection\n" );
  fprintf( stdout, "     parameters: collection name, field name, value\n");
  fprintf( stdout, "\n");
  fprintf( stdout, "(3)  [-listIds] to list all object ids of a collection\n" );
  fprintf( stdout, "     parameters: collection name\n");
  fprintf( stdout, "\n");
  fprintf( stdout, "(4)  [-listSelectionNames] to list the selection name of\n" );
  fprintf( stdout, "                           all objects of a collection\n" );
  fprintf( stdout, "     parameters: collection name\n");
  fprintf( stdout, "\n");
  fprintf( stdout, "(5)  [-deleteCollection]   to delete a collection\n");
  fprintf( stdout, "     parameters: collection name\n");
  fprintf( stdout, "\n");
  fprintf( stdout, "(6)  [-isnodeincoll]  to check to see if a given node is\n");
  fprintf( stdout, "                      in a named collection.\n");
  fprintf( stdout, "     parameters: node name, collection name\n");
  fprintf( stdout, "========================================================================\n" );
}
```

Figure 210 (Part 1 of 7). Sample Program Using the Collection API, wtcoll.c

```
/*
   Database manipulation functions
*/

int createObjectField( OVwObjectId objectId,
                 char *fieldName,
                 char *value )
{
  OVwFieldId     newfieldId;

  /* Create the database field */
  newfieldId = OVwDbCreateField( fieldName, ovwStringField, ovwGeneralField );

  if ( newfieldId == -1 )   /* Could not create field */
    return -1;

  /* Set the database field */
  if ( OVwDbSetFieldStringValue( objectId, newfieldId, value ) == -1 )
    return NULL;
}

char *getObjectField( OVwObjectId objectId,
                char *fieldName )
{
  OVwFieldId    fieldId;

  fieldId = OVwDbFieldNameToFieldId( fieldName );

  if ( (fieldId == ovwNullFieldId) ||
       (objectId == -1) )
    return NULL;

  return OVwDbGetFieldStringValue( objectId, fieldId );
}


OVwObjectId setObjectField( OVwObjectId objectId,
                   char *fieldName,
                   char *value )
{
  OVwFieldId     fieldId;

  fieldId = OVwDbFieldNameToFieldId(fieldName);

  if ( (fieldId == ovwNullFieldId) ||
       (objectId == -1) )
    return NULL;

  if ( OVwDbSetFieldStringValue( objectId, fieldId, value ) == -1 )
    return NULL;

  return objectId;
}

/*
   Collection based functions
*/

void createFieldOnObject( char *collectionName, char *selectionName,
                 char *fieldName     , char *value )
{
  int  count;
  int  collectionConnectionFD; /* collection connection file descriptor */

  OVwObjectIdList *objectList;
  OVwObjectId objectId, tmpObjectId;

  /* Estabilishes a connection with the Collecion Facility daemon */
  collectionConnectionFD = nvCollectionOpen();

  fprintf( stderr, "selection Name: %s\n", selectionName );
```

Figure 210 (Part 2 of 7). Sample Program Using the Collection API, wtcoll.c

```
  /* Verifies if the collection has been sucessfully created */
  if ( collectionConnectionFD == -1 )
    fprintf( stderr, "Unsucessfull operation: %s\n",
          nvCollectionErrorMsg(nvCollectionError()));
  else
    {
      objectId = OVwDbSelectionNameToObjectId( selectionName );

      /* get a list of objects of a collection */
      objectList = nvCollectionResolve( collectionName );

      for ( count = 0 ; count < objectList->count ; count++ )
     {
       tmpObjectId = objectList->object_ids[count];

       if ( tmpObjectId == objectId )
         if ( setObjectField( objectId, fieldName, value ) == NULL )
           {
          if ( createObjectField ( objectId, fieldName, value ) == NULL )
            fprintf( stderr, "Could not create object field!\nSelection name: %s\n", selectionName );
          break;
           }
     }

      /* Free the object list */
      OVwDbFreeObjectIdList( objectList );
    }

  nvCollectionDone();
}


void setFieldOnAllObjects( char *collectionName, char *fieldName, char *value )
{
  char *selectionName;

  int  count;
  int  collectionConnectionFD; /* collection connection file descriptor */

  OVwObjectIdList *objectList;
  OVwObjectId objectId;

  /* Estabilishes a connection with the Collecion Facility daemon */
  collectionConnectionFD = nvCollectionOpen();

  /* Verifies if the collection has been sucessfully created */
  if ( collectionConnectionFD == -1 )
    fprintf( stderr, "Unsucessfull operation: %s\n",
          nvCollectionErrorMsg(nvCollectionError()));
  else
    {
      /* get a list of objects of a collection */
      objectList = nvCollectionResolve( collectionName );

      for ( count = 0 ; count < objectList->count ; count++ )
     {
       objectId = objectList->object_ids[count];
       fprintf( stdout, "%d\n", objectId );

       selectionName = OVwDbObjectIdToSelectionName(objectId);
       if ( setObjectField( objectId, fieldName, value ) == NULL )
         {
           if ( createObjectField ( objectId, fieldName, value ) == NULL )
          fprintf( stderr, "Could not create object field!\nSelection name: %s\n", selectionName );
         }
```

Figure 210 (Part 3 of 7). Sample Program Using the Collection API, wtcoll.c

```
       if ( selectionName )
         free( selectionName );
     }

      /* Free the object list */
      OVwDbFreeObjectIdList( objectList );
    }

  nvCollectionDone();
}

void listAllObjects( char *collectionName, int IdOrSelectionName )
{
  char *selectionName;

  int  count;
  int  collectionConnectionFD; /* collection connection file descriptor */

  OVwObjectIdList *objectList;
  OVwObjectId objectId;

  /* Estabilishes a connection with the Collecion Facility daemon */
  collectionConnectionFD = nvCollectionOpen();

  /* Verifies if the collection has been sucessfully created */
  if ( collectionConnectionFD == -1 )
    fprintf( stderr, "Unsucessfull operation: %s\n",
          nvCollectionErrorMsg(nvCollectionError()));
  else
    {
      /* get a list of objects of a collection */
      objectList = nvCollectionResolve( collectionName );

      for ( count = 0 ; count < objectList->count ; count++ )
     {
       objectId = objectList->object_ids[count];

       if ( IdOrSelectionName == ID )
         fprintf( stdout, "%d\n", objectId );
       else
         {
           selectionName = OVwDbObjectIdToSelectionName(objectId);
           fprintf( stdout, "%s\n", selectionName );

           if ( selectionName )
          free( selectionName );
         }
     }

      /* Free the object list */
      OVwDbFreeObjectIdList( objectList );
    }

  nvCollectionDone();
}


void deleteCollection( char *collectionName )
{
  char *selectionName;

  int  collectionFD;           /* collection file descriptor */
  int  collectionConnectionFD; /* collection connection file descriptor */

  /* Estabilishes a connection with the Collecion Facility daemon */
  collectionConnectionFD = nvCollectionOpen();
```

Figure 210 (Part 4 of 7). Sample Program Using the Collection API, wtcoll.c

```
  /* Verifies if the collection has been sucessfully created */
  if ( collectionConnectionFD == -1 )
    fprintf( stderr, "Unsucessfull operation: %s\n",
          nvCollectionErrorMsg(nvCollectionError()) );
  else
    {
      /* Delete the new collection to the collections list */
      /* It may take some CPU processing if the collection is large */
      collectionFD = nvCollectionDelete( collectionName , 1 );
    }

  nvCollectionDone();
}

int goexit(int retc) {

  /* close the collection connection and exit */
  nvCollectionDone() ;
  exit(retc) ;
}

void isNodeInCollection(char *node_name, char *coll_name) {

  OVwObjectId         node_objid ;
  nvCollectionList  * coll_list ;
  nvCollectionDefn  * list_entry ;
  int                 collectionConnectionFD, i;

  /* Establish the connection to the collection facility */
  if ((collectionConnectionFD = nvCollectionOpen()) < 0) {
       fprintf(stderr, "CollectionOpen failed. %s\n",
               nvCollectionErrorMsg(nvCollectionError())) ;
       exit(-1) ;
       }

  /* Open the ovwdb API */
  if (OVwDbInit() == EOF) {
       fprintf(stderr, "OVwDbInit failed: %s\n", OVwErrorMsg(OVwError())) ;
       goexit(-1) ;
       }

  /* Get the object ID of the node */
  if ((node_objid = OVwDbSelectionNameToObjectId( node_name )) == NULL) {
       fprintf(stderr, "Error obtaining object ID: %s\n", OVwErrorMsg(OVwError())) ;
       goexit(-1) ;
       }

  /* Get a list of collections that the object is in */
  if ((coll_list = nvCollectionGetAllForObject( node_objid )) == NULL) {
       fprintf(stderr, "Error getting collections for node: %s\n",
               nvCollectionErrorMsg(nvCollectionError())) ;
       goexit(-1) ;
       }

  /* Loop down the list checking for matches with the requested collection */
  list_entry = coll_list->collections ;
  for (i=0; i < coll_list->count ; i++) {
      if ((strcmp(coll_name, list_entry++->name)) == 0 ) {
         printf("%s is in collection %s\n", node_name, coll_name) ;
         goexit(1) ;
         }
      }
  printf("%s is not in collection %s\n", node_name, coll_name) ;
  goexit(-1) ;
}
```

Figure 210 (Part 5 of 7). Sample Program Using the Collection API, wtcoll.c

```
/*
   main function
*/
void main( int argc , char **argv )
{
  /*
     collectionName - used to refer to the new collection
  */

  char *collectionName;
  char *fieldName;
  char *value;
  char *selectionName;
  char *option;

  if ( argc-- < 3 )
    {
      usage();
      exit(0);
    }

  argv++;

  OVwDbInit();

  map = OVwGetMapInfo();

  option = *argv++;
  argc--;

  if ( !strcmp(option, "-setField") )
    {
      if ( argc != 3 )
     {
       usage();
       exit(0);
     }

      collectionName = *argv++;
      fieldName = *argv++;
      value = *argv++;

      setFieldOnAllObjects( collectionName, fieldName, value );
    }
  else
    if ( !strcmp(option, "-createField") )
      {
     if ( argc != 4 )
       {
         usage();
         exit(0);
       }

     selectionName = *argv++;
     collectionName = *argv++;
     fieldName = *argv++;
     value = *argv++;

     createFieldOnObject( collectionName, selectionName, fieldName, value );
      }
    else
      if ( !strcmp(option, "-listIds") )
     {
       if ( argc != 1 )
         {
           usage();
           exit(0);
         }

       collectionName = *argv++;
       listAllObjects( collectionName, ID );
     }
```

Figure 210 (Part 6 of 7). Sample Program Using the Collection API, wtcoll.c

```
      if ( !strcmp(option, ″-listSelectionNames″) )
        {
          if ( argc != 1 )
            {
           usage();
           exit(0);
            }

          collectionName = *argv++;
          listAllObjects( collectionName, SELECTION_NAME );
        }
      else
        if ( !strcmp(option, ″-deleteCollection″) )
          {
            if ( argc != 1 )
           {
             usage();
             exit(0);
           }

            collectionName = *argv++;
            deleteCollection( collectionName );
          }
        else
          if ( !strcmp(option, ″-isnodeincoll″) )
            {
           if ( argc != 2 )
             {
               usage();
               exit(0);
             }

                 selectionName = *argv++;
             isNodeInCollection( selectionName, *argv );
            }
}
```

Figure 210 (Part 7 of 7). Sample Program Using the Collection API, wtcoll.c

Appendix C. C Code for the wtdepend_list Sample Program

```
/*---------------------------------------------------------------------
  wtdepend_list.c

  This program maintains a collection of collections called
  unreachable_nodes.  It is created and maintained by the wtdepend.rs
  ruleset. The unreachable_nodes collection is then referenced by the
  wtdepdisp.rs ruleset to check if a given node is reachable or not.

  This program is a sample from ITSO Redbook, "Examples Using NetView for AIX
  Version 4"(SG24-4515).

  Author: Rob Macgregor, ITSO-Raleigh

  Copyright (C) IBM Corporation 1995
  ---------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <OV/ovw.h>
#include <OV/nvCollection.h>
#include <OV/nvCollectionErrs.h>

int      collectionConnectionFD ;

int goexit(int retc) {
/* close the collection connection and exit */
nvCollectionDone() ;
exit(retc) ;
}


int main(int argc, char **argv){

char             * coll_1_name = "unreachable_nodes" ;
char             * coll_2_name ;
char               * coll_desc ;
char             * coll_rule ;
char             * action ;
/* This is not nice. Should re-write with dynamic space allocation (sometime...) */
char             * rule_list[40] ;
char               new_rule[1024] ;

char             * rule_type ;
int                i, errcode, rule_count=0, remaining=0  ;

if ( argc != 3 )
   {
    printf("Usage: wtdepend_list collection action\n") ;
    exit(-1) ;
   }

coll_2_name=*++argv ;
action=*++argv ;
printf("you said %s %s\n", action, coll_2_name) ;
```

Figure 211 (Part 1 of 3). Sample Program to Maintain a Collection of Collections, wtdepend_list.c

```
/* Establish the connection to the collection facility */
if ((collectionConnectionFD = nvCollectionOpen()) < 0) {
     printf("CollectionOpen failed. %s\n",
             nvCollectionErrorMsg(nvCollectionError())) ;
     exit(-1) ;
     }


/* Get the rule for the current list of unreachable nodes */
if (nvCollectionGetInfo( coll_1_name, &coll_desc, &coll_rule ) != 0) {
     errcode = nvCollectionError() ;
     if ( (errcode == NV_COLLECTION_DOES_NOT_EXIST ) &&
          (strcmp(action, "ADD" ) == 0)) {
          /* Create the unreachable node collection */
          strcpy(new_rule, "(IN_COLLECTION ") ;
          strcat(new_rule, coll_2_name) ;
          strcat(new_rule, ")" ) ;
          printf("Will create %s with rule: %s\n", coll_1_name, new_rule) ;
          if (nvCollectionAdd( coll_1_name,
                              "Collection of dependent node collections",
                              new_rule, FALSE) != 0) {
               printf("Error adding %s collection: %s\n", coll_1_name, \
                             nvCollectionErrorMsg(nvCollectionError())) ;
               goexit(-1) ;
               }
          printf("Added collection %s\n", coll_1_name) ;
          goexit(0) ;
          }

     else {
          printf("Error getting collection information: %s\n",
                  nvCollectionErrorMsg(errcode)) ;
          goexit(-1) ;
          }
     }

else {

      /* Collection exists. Find all the pieces */

      rule_type = (char *) strtok(coll_rule, " ()|" ) ;

      while ((rule_list[rule_count++] = (char *) strtok(NULL, " ()|" )) != NULL) {
          if( strcmp(rule_type, "IN_COLLECTION") != 0) {
              printf("Program cannot deal with this kind of collection\n") ;
              exit(-1) ;
              }
          rule_type = (char *) strtok(NULL , " ()|" ) ;
          }
      rule_count -= 2 ;
      }

/* Now handle the ADD or DEL function */
if (strcmp(action, "ADD") == 0) {
    strcpy(new_rule, "((IN_COLLECTION ") ;

    for(i=0 ; i <= rule_count ; i++) {
       strcat(new_rule, rule_list[i]) ;
       strcat(new_rule, ") || (IN_COLLECTION ") ;
       }
    strcat(new_rule, coll_2_name) ;
    strcat(new_rule, "))" ) ;
    /* update the rule */
    if (nvCollectionModify( coll_1_name,
                           "Collection of dependent node collections",
                            new_rule, FALSE) != 0) {
               printf("Error modifying %s collection: %s\n", coll_1_name, \
                             nvCollectionErrorMsg(nvCollectionError())) ;
               goexit(-1) ;
               }
     goexit(0) ;
     }
```

Figure 211 (Part 2 of 3). Sample Program to Maintain a Collection of Collections, wtdepend_list.c

```
if (strcmp(action, "DEL") == 0) {
    strcpy(new_rule, "((IN_COLLECTION ") ;

    remaining = rule_count + 1 ;
    for(i=0 ; i <= rule_count ; i++) {
       if(strcmp(coll_2_name, rule_list[i]) != 0) {
           remaining -= 1 ;
           strcat(new_rule, rule_list[i]) ;
           if(remaining > 0) strcat(new_rule, ") || (IN_COLLECTION ") ;
           }
       else if(rule_count == 0) {
           nvCollectionDelete(coll_1_name) ;
           exit(0) ;
           }
       }
    strcat(new_rule, "))" ) ;
    printf("%s\n", new_rule) ;
    /* update the rule */
    if (nvCollectionModify( coll_1_name,
                           "Collection of dependent node collections",
                            new_rule, FALSE) != 0) {
               printf("Error modifying %s collection: %s\n", coll_1_name, \
                           nvCollectionErrorMsg(nvCollectionError())) ;
               goexit(-1) ;
               }
     goexit(0) ;
     }

}
```

Figure 211 (Part 3 of 3). Sample Program to Maintain a Collection of Collections, wtdepend_list.c

Appendix D. C Code for the wtotapi1 Sample Program

The wtotapi1 utility has been extensively modified and improved since its first appearence in *Examples of Using NetView for AIX*, GG24-4327. It now supports various command line options and an improved user interface. The source for wtotapi1 (including debuging information) is listed in Figure 214 on page 272.

The usage information (which can be obtained using command: `wtotapi -?`) is as follows:

```
wtotapi1 [ -s <server> ] [ -h ] [ -p ]
[ -o <OID> ] [ -P <protocol> ]
[ -f <submap> ] [ -c <command> ]
```

Figure 212. wtotapi1 Usage

-s <server> Indicates the server on which the gtmd is running.

-h Displays the help message as shown in Figure 213.

-p Will turn off the ″Command>″ prompt, useful when redirecting commands from input files.

-o <OID> Will set the oid prefix, equivalent to `prefix` command from the user interface.

-P <protocol> Will set the protocol number, equivalent to the `protocol` command from the user interface.

-f <submap> Will set the focus, equivalent to the `focus` command from the user interface.

-c <command> Will instigate wtotapi1 in command line mode. The command (which must be one argument only) will be executed and then wtotapi1 will exit.

```
Commands:

protocol <Name>                     Set protocol number to ′Name′
parent <Name>                       Set parent to ′Name′
prefix <Name>                       Set prefix to ′Name′
focus <Name>                        Set graph to ′Name′
add graph <Name> <Layout> <Type>    Add graph ′Name′ to current graph
add box <Name> <Type> <Layout>      Add box-graph ′Name′ to current graph
add vertex <Name> <Type>            Add vertex ′Name′ to the current graph
add arc <Name> <Name>               Add connection ′Name′ between
                                    box graphs or vertexes
add sap p <Vname>                   Add a providing SAP ′Vname′
add sap u <Vname> <Vname> <P>       Add a ′using′ SAP ′Vname′
set <Vertname> <Status>             Change the status of a vertex
help                                Print this message
quit                                Quit Application
```

Figure 213. wtotapi1 Help Message

```
/****************************************************************\
******************************************************************
  Sample Program to drive the NV6000 V3 gtm API

  AUTHOR Rob Macgregor
  HISTORY
  Improvments to fix bugs and add functionality - Mark J Hodge
******************************************************************
\****************************************************************/

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <wordexp.h>
#include <unistd.h>

#include <OV/ovw_obj.h>
#include <nvot.h>

#define  MAX_NAME_LEN 48

typedef enum gtm_cmd_id {
  GTM_SET_PROTOCOL,        /* Set current protocol ID   */
  GTM_SET_PREFIX,        /* Set prefix of protocol oid*/
  GTM_SET_PARENT,        /* Set parent graph          */
  GTM_ADD_OBJECT,        /* Add something to network  */
  GTM_DEL_OBJECT,        /* Remove something          */
  GTM_SET_OBJECT,        /* Set variable for object   */
  GTM_QUIT,            /* Set variable for object   */
  GTM_ADD_GRAPH,        /* Add graph command         */
  GTM_ADD_BOX,            /* Add box graph command     */
  GTM_ADD_VERTEX,        /* Add vertex                */
  GTM_ADD_ARC,            /* Add arc                   */
  GTM_ADD_SAP,            /* Add sap                   */
  HELP,
  QUIT
} gtm_cmd_id;

/* Global variables - current protocol, current graph name and graph protcol */

char                cmd_string[80] ;
char                        oid_prefix[40] = "1.3.6.1.2.1.2.2.1.3" ;
char                        current_prot_char[40] ;
nvotGraphProtocolType       current_prot_oid = current_prot_char ;
nvotVertexProtocolType      current_prot_nbr = 0 ;
char                        current_graph[32] ;
char                        current_graph_prot[40] ;
nvotGraphType               current_graph_type ;

extern int optind;
extern char *optarg;

int   ep_to_arc_binding[] = {0,0,
                   ARC_GRAPH_GRAPH_NAME_BINDING,
                   ARC_VERTEX_GRAPH_NAME_BINDING,
                   ARC_GRAPH_VERTEX_NAME_BINDING,
                   ARC_VERTEX_VERTEX_NAME_BINDING} ;


void usage()
{
  printf("Usage: wtotapi1 [ -s <server> ] [ -h ] [ -p ] [ -o <OID> ] [ -P
<protocol> ] [ -f <submap> ] [ -c <command> ]\n");
}
```

Figure 214 (Part 1 of 12). wtotapi1.c Source Code (Including Debug Information)

```
int print_help()
{
  printf(″Commands:\n\n″);
  printf(″protocol <Name>\t\t\t\tSet protocol number to ′Name′\n″);
  printf(″parent <Name>\t\t\t\tSet parent to ′Name′\n″ );
  printf(″prefix <Name>\t\t\t\tSet prefix to ′Name′\n″ );
  printf(″focus <Name>\t\t\t\tSet graph to ′Name′\n″);
  printf(″add graph <Name> <Layout> <Type>\tAdd graph ′Name′ to current graph\n″);
  printf(″add box <Name> <Type> <Layout>\t\tAdd box-graph ′Name′ to current graph\n″);
  printf(″add vertex <Name> <Type>\t\tAdd vertex ′Name′ to the current graph\n″);
  printf(″add arc <Name> <Name>\t\t\tAdd connection ′Name′ between\n\t\t\t\t\tbox graphs or
vertexes\n″);
  printf(″add sap p <Vname>\t\t\tAdd a providing SAP ′Vname′\n″);
  printf(″add sap u <Vname> <Vname> <P>\t\tAdd a ′using′ SAP ′Vname′\n″);
  printf(″set <Vertname> <Status>\t\t\tChange the status of a vertex\n″);
  printf(″help\t\t\t\t\tPrint this message\n″ );
  printf(″quit\t\t\t\t\tQuit Application\n″ );
  return( 0 );
}

/* Read /usr/OV/conf/C/oid_to_sym and return OID for given Icon name */
int sym_to_oid(char *icon, char *oid)
{
  FILE *fid ;
  char instring[120] ;
  char *mapping_string ;
  char *end_ptr ;
  char *dotted_decimal_oid ;
  char *sym_name ;
  char *oid_to_sym_file = ″/usr/OV/conf/C/oid_to_sym″ ;

  if ((fid = fopen(oid_to_sym_file, ″r″)) == NULL) {
    printf(″Error reading oid_to_sym file″) ;
    exit(-1) ;
  }
  while( fgets(instring, 120, fid) != 0) {
    if (strspn( instring, ″#″) == 0 ) {
      mapping_string = strtok(instring, ″#″) ;
    /* Strip off trailing blanks and tabs */
      for (
       end_ptr = mapping_string + strlen(mapping_string) -1 ;
       (strcmp(end_ptr, ″ ″) == 0) || (strcmp(end_ptr, ″\t″) == 0) ;
       strcpy(end_ptr, 0x00), end_ptr--
       ) ;

      if (strlen(mapping_string) > 1 ) {
    dotted_decimal_oid = strtok(mapping_string, ″:″) ;
    sym_name = strtok(NULL, ″\n″) ;
        if(strncmp(oid_prefix, dotted_decimal_oid, strlen(oid_prefix) ) == 0 ) {
          if(strcmp(sym_name, icon) == 0 ) {
        strcpy( oid, dotted_decimal_oid );
#ifdef _DEBUG
            printf( ″sym_to_oid: oid_prefix\t= %s\n″, oid_prefix );
            printf( ″sym_to_oid: dotted_decimal_oid\t= %s\nsym_to_oid: sym_name\t= %s\n″,
dotted_decimal_oid, sym_name );
            printf( ″sym_to_oid: oid\t\t= %s\nsym_to_oid: sym_name\t= %s\n″, oid, sym_name );
#endif
            return( 0 );
      }
        }
      }
    }
  }
  printf(″No entry for %s found in %s. Try again\n″, icon, oid_to_sym_file) ;
  return( -1 );
}
```

Figure 214 (Part 2 of 12). wtotapi1.c Source Code (Including Debug Information)

```
/* Routine to qualify what command was issued */
int cmd_to_table(char cmd[10] )
{
  struct cmdtable {
    char *c_name;
    int c_desc;
  } cmdtable[] = {
    { "protocol",   GTM_SET_PROTOCOL },
    { "prot",       GTM_SET_PROTOCOL },
    { "parent",     GTM_SET_PARENT },
    { "focus",      GTM_SET_PARENT },
    { "prefix",     GTM_SET_PREFIX },
    { "add",        GTM_ADD_OBJECT },
    { "a",          GTM_ADD_OBJECT },
    { "set",        GTM_SET_OBJECT },
    { "s",          GTM_SET_OBJECT },
    { "graph",      GTM_ADD_GRAPH },
    { "vertex",     GTM_ADD_VERTEX },
    { "box",        GTM_ADD_BOX },
    { "arc",        GTM_ADD_ARC },
    { "sap",        GTM_ADD_SAP },
    { "h",          HELP },
    { "?",          HELP },
    { "help",       HELP },
    { "q",          QUIT },
    { "quit",       QUIT },
    { NULL,         EOF }
  } ;
  int i ;

  for(i=0; cmdtable[i].c_name != NULL; i++)
    if(!strcmp(cmdtable[i].c_name, cmd))
      return(cmdtable[i].c_desc);
  return(EOF) ;
}

/* Routine to qualify what status is required */
int char_to_status(char *char_status)
{
  struct status_tbl {
    char *s_name;
    int s_desc;
  } status_tbl[] = {
    { "up",       STATUS_NORMAL },
    { "down",     STATUS_CRITICAL },
    { "marginal", STATUS_MARGINAL },
    { "marg",     STATUS_MARGINAL },
    { "unknown",  STATUS_UNKNOWN },
    { "unk",      STATUS_UNKNOWN },
    { NULL,         EOF }
  } ;
  int i ;
  for(i=0; status_tbl[i].s_name != NULL; i++)
    if(!strcmp(status_tbl[i].s_name, char_status))
      return(status_tbl[i].s_desc);
  printf("Invalid status, options are: up, down, marginal, unknown\n");
  return(EOF) ;
}
/* Set the protocol number we are currently working with */
int set_cur_protocol()
{
  if( (current_prot_nbr = atoi(cmd_string)) == NULL)
    {
      printf("The protocol id has to be a number\n") ;
      return( -1 );
    }
  strcpy(current_prot_oid, oid_prefix) ;
  strcat(current_prot_oid, ".") ;
  strcat(current_prot_oid, cmd_string) ;
  printf("Successfull operation. Protocol = %d\n", current_prot_nbr) ;
  return( 0 );
}
```

Figure 214 (Part 3 of 12). wtotapi1.c Source Code (Including Debug Information)

```
/* Set the graph. All ″add″s subsequently will be members of the
   graph defined in here */
int set_cur_graph()
{
  char * graph_name = cmd_string ;
  /* We must have set a value for current protocol */
  if (current_prot_nbr == 0)
    {
      printf(″You have to select a protocol ID before\n″) ;
      printf(″You can set the parent graph\n″) ;
      return( -1 );
    }
  /* We want to know if it exists, and whether it is a box or graph-graph*/
  nvotGetVerticesInGraph(current_prot_oid, graph_name) ;
  if (nvotGetError() == NVOT_SUCCESS ) current_graph_type = GRAPH ;
  else {
    nvotGetVerticesInBox(current_prot_oid, graph_name) ;
    if (nvotGetError() == NVOT_SUCCESS ) current_graph_type = BOX ;
    else {
      printf (″%s is not a box-graph or graph-graph in protocol %d\n″,
          graph_name, current_prot_nbr) ;
      return( -1 );
    }
  }

  strcpy(current_graph, graph_name) ;
  strcpy(current_graph_prot, current_prot_oid) ;
  printf(″Successfull operation. Current Graph = %s\n″, current_graph) ;
  return( 0 );
}

/* Convert layout character to nvotLayout value */
nvotLayoutType char_to_layout(char * layout_string)
{
  struct layout_tbl{
    char *l_name;
    nvotLayoutType l_type;
  } layout_tbl[] = {

    { ″none″,    NONE_LAYOUT },
    { ″p2p″,     POINT_TO_POINT_LAYOUT },
    { ″bus″,     BUS_LAYOUT },
    { ″star″,    STAR_LAYOUT },
    { ″ring″,    SPOKED_RING_LAYOUT },
    { ″rowcol″,  ROWCOL_LAYOUT },
    { ″p2pring″, POINT_TO_POINT_RING_LAYOUT },
    { ″tree″,    TREE_LAYOUT },
    { NULL,      NONE_LAYOUT }
  };
  int i ;
  for(i=0; layout_tbl[i].l_name != NULL; i++)
    if(!strcmp(layout_tbl[i].l_name, layout_string))
      return(layout_tbl[i].l_type);
  printf(″Unknown layout type - NONE_LAYOUT will be used\n″) ;
  return(NONE_LAYOUT) ;
}

/* Add graph to Root map  */
int add_root_graph()
{
  char   *tstr ;
  char   icon_oid[40] ;
  char   graph_name[MAX_NAME_LEN]  ;
  char   graph_icon[30] ;
  char   graph_layout_c[10] ;
  nvotLayoutType  graph_layout ;
  nvotOctetString graph_details ;
  OVwObjectId objid ;

  strcpy( graph_name, strtok( NULL, ″ ″ )) ;
  strcpy( graph_icon, strtok(NULL, ″ ″ )) ;
  strcpy( graph_layout_c, strtok( NULL, ″ ″ )) ;
```

Figure 214 (Part 4 of 12). wtotapi1.c Source Code (Including Debug Information)

```
  while ( tstr = strtok(NULL, " " ))
      {
        strcat(graph_icon, " " ) ;
        strcat(graph_icon, graph_layout_c) ;
    strcpy(graph_layout_c, tstr);
      }
#ifdef _DEBUG
  printf("add_root_graph: graph_name\t= %s\nadd_root_graph: graph_icon\t= %s\nadd_root_graph:
graph_layout_c\t= %s\n", graph_name, graph_icon, graph_layout_c );
#endif

  if (sym_to_oid( graph_icon, &icon_oid) != 0)
    return( -1 );
  graph_layout = char_to_layout(graph_layout_c) ;
  objid = nvotCreateRootGraph(
                 current_prot_oid,
                 &graph_name,
                 graph_layout,
                 "",
                 &icon_oid,
                 &graph_name,
                 &graph_details) ;
  printf("%s", nvotGetErrorMsg(nvotGetError())) ;
  printf(" Object = %d\n", objid) ;
  return( 0 );
}

/* Add graph graph  */
int add_graph_graph()
{
  char   *tstr ;
  char   icon_oid[40] ;
  char   graph_name[MAX_NAME_LEN]  ;
  char   graph_icon[30] ;
  char   graph_layout_c[10] ;
  nvotLayoutType  graph_layout ;
  nvotOctetString graph_details ;
  OVwObjectId objid ;

  strcpy( graph_name, strtok( NULL, " " )) ;
  strcpy( graph_icon, strtok(NULL, " " )) ;
  strcpy( graph_layout_c, strtok( NULL, " " )) ;
  while ( tstr = strtok(NULL, " " ))
      {
        strcat(graph_icon, " " ) ;
        strcat(graph_icon, graph_layout_c );
        strcpy(graph_layout_c, tstr) ;
      }
#ifdef _DEBUG
  printf("add_graph_graph: graph_name\t= %s\nadd_graph_graph: graph_icon\t= %s\nadd_graph_graph:
graph_layout_c\t= %s\n", graph_name,
graph_icon, graph_layout_c );
#endif

  if (sym_to_oid( graph_icon, &icon_oid) != 0)
    return( -1 );
  graph_layout = char_to_layout(graph_layout_c) ;
  objid = nvotCreateGraphInGraph(
                &current_graph_prot,
                &current_graph,
                current_prot_oid,
                &graph_name,
                    graph_layout,
                "",
                &icon_oid,
                &graph_name,
                &graph_details) ;
  printf("%s\n", nvotGetErrorMsg(nvotGetError())) ;
  return( 0 );
}
```

Figure 214 (Part 5 of 12). wtotapi1.c Source Code (Including Debug Information)

```
/* Add box graph  */
int add_box_graph(args)
{
  char   *tstr ;
  char   icon_oid[40] ;
  char   graph_name[MAX_NAME_LEN]  ;
  char   graph_icon[30] ;
  char   graph_layout_c[10] ;
  nvotLayoutType  graph_layout ;
  nvotOctetString graph_details ;
  OVwObjectId objid ;
  strcpy( graph_name, strtok( NULL, " " )) ;
  strcpy( graph_icon, strtok(NULL, " " )) ;
  strcpy( graph_layout_c, strtok( NULL, " " )) ;
  while ( tstr = strtok(NULL, " " ))
      {
        strcat(graph_icon, " " ) ;
    strcat(graph_icon, graph_layout_c );
        strcpy(graph_layout_c, tstr) ;
      }
#ifdef _DEBUG
  printf("add_box_graph: graph_name\t= %s\nadd_box_graph: graph_icon\t= %s\nadd_box_graph:
graph_layout_c\t= %s\n", graph_name,
graph_icon, graph_layout_c );
#endif
  if (sym_to_oid( graph_icon, &icon_oid) != 0)
    return( -1 );
  graph_layout = char_to_layout(graph_layout_c) ;
  objid = nvotCreateBoxInGraph(
                  &current_graph_prot,
                  &current_graph,
                  current_prot_oid,
                  &graph_name,
                  graph_layout,
                  "",
                  &icon_oid,
                  &graph_name,
                  &graph_details) ;
  printf("%s", nvotGetErrorMsg(nvotGetError())) ;
  printf(" Object = %d\n", objid) ;
  return( 0 ) ;
}

/* Add vertex     */
int add_vertex(args)
{
  char   *tstr ;
  char   icon_oid[40] ;
  char   vertex_name[MAX_NAME_LEN]  ;
  char   vertex_icon[30] ;
  nvotOctetString vertex_details ;
  OVwObjectId objid ;
  strcpy( vertex_name, strtok( NULL, " " )) ;
  strcpy( vertex_icon, strtok(NULL, " " )) ;
  while ( tstr = strtok(NULL, " " ))
      {
        strcat(vertex_icon, " " ) ;
        strcat(vertex_icon, tstr) ;
      }
#ifdef _DEBUG
 printf("add_vertex: vertex_name\t= %s\nadd_vertex: vertex_icon\t= %s\n", vertex_name, vertex_icon );
#endif
  if (sym_to_oid( vertex_icon, &icon_oid) != 0)
    return( -1 ) ;
```

Figure 214 (Part 6 of 12). wtotapi1.c Source Code (Including Debug Information)

```
  if(current_graph_type == BOX)
    objid = nvotCreateVertexInBox(
                  &current_graph_prot,
                  &current_graph,
                  current_prot_nbr,
                  &vertex_name,
                  &icon_oid,
                  &vertex_name,
                  &vertex_details,
                  STATUS_NORMAL) ;
  else objid = nvotCreateVertexInGraph(
                       &current_graph_prot,
                       &current_graph,
                       current_prot_nbr,
                       &vertex_name,
                       &icon_oid,
                       &vertex_name,
                       &vertex_details,
                       STATUS_NORMAL) ;
  printf("%s", nvotGetErrorMsg(nvotGetError())) ;
  printf("Object = %d\n", objid) ;
  return( 0 ) ;
}

/* Add an arc     */
int add_arc()
{
  char          end_pt_a[MAX_NAME_LEN] ;
  char          end_pt_z[MAX_NAME_LEN] ;
  char             arcid_char[4] ;
  int              arcid  = 0 ;
  int              end_type_a ;
  int              end_type_z ;
  OVwObjectId      objid ;
  nvotProtocolType end_prot_a ;
  nvotProtocolType end_prot_z ;

  strcpy(&end_pt_a, strtok( NULL, " " )) ;
  strcpy(&end_pt_z, strtok( NULL, " " )) ;

  /* the end point may be vertices or graphs. Test to see which*/
  nvotGetVerticesInBox(current_prot_oid, end_pt_a) ;
  if(nvotGetError() == NVOT_SUCCESS) {
    end_type_a = 0 ;
    end_prot_a.graphProtocol = current_prot_oid ;
  } else {
    end_type_a = 1 ;
    end_prot_a.vertexProtocol = current_prot_nbr ;
  }

  nvotGetVerticesInBox(current_prot_oid, end_pt_z) ;
  if(nvotGetError() == NVOT_SUCCESS) {
    end_type_z = 2 ;
    end_prot_z.graphProtocol = current_prot_oid ;
  } else {
    end_type_z = 4 ;
    end_prot_z.vertexProtocol = current_prot_nbr ;
  }
  if (current_graph_type == GRAPH)
    {
      objid = nvotCreateArcInGraph ( &current_graph_prot,
                    &current_graph,
                    ep_to_arc_binding[end_type_a + end_type_z],
                    end_prot_a,
                    &end_pt_a,
                    end_prot_z,
                    &end_pt_z,
                    arcid,
                    NULL     ,
                    NULL,
                    NULL,
                    STATUS_NORMAL) ;
```

Figure 214 (Part 7 of 12). wtotapi1.c Source Code (Including Debug Information)

```
       printf("%s", nvotGetErrorMsg(nvotGetError())) ;
       printf("Object = %d\n", objid) ;
    } else {
       printf("Arcs can only be added to a Graph graph, %s is a Box graph\n",
          current_graph) ;
       return( -1 ) ;
    }
  return( 0 );
}

/* Add a providing or using SAP */
int add_sap()
{
  char      sap_type[9] ;
  char       sap_user_name[MAX_NAME_LEN] ;
  char      sap_provider_name[MAX_NAME_LEN] ;
  nvotVertexProtocolType      sap_provider_prot ;

  strcpy(sap_type, strtok( NULL, " " )) ;
  if (sap_type[0] == 'p') {    /* a "Providing" SAP */
    strcpy(&sap_provider_name, strtok( NULL, " " )) ;
    printf("About to add providing SAP entry %s, protocol %d\n",
       sap_provider_name, current_prot_nbr) ;
    nvotCreateProvidingSap(
                  current_prot_nbr,
                  &sap_provider_name,
                  77,
                  &sap_provider_name) ;    /* Note: uses vertex name as SAP name*/
    printf("%s\n", nvotGetErrorMsg(nvotGetError())) ;
    return( 0 ) ;
  }
  if (sap_type[0] == 'u') {    /* a "Using" SAP */
    strcpy(&sap_user_name, strtok(NULL, " ")) ;
    strcpy(&sap_provider_name, strtok(NULL, " ")) ;
    sap_provider_prot = atoi(strtok(NULL, " ")) ;
    printf("About to add using SAP entry %s, to SAP provided by %s protocol %d\n",
       sap_user_name, sap_provider_name, sap_provider_prot) ;
    nvotCreateUsingSap(
                  current_prot_nbr,
                  &sap_user_name,
                  77,
                  &sap_provider_name) ;
    printf("%s\n", nvotGetErrorMsg(nvotGetError())) ;
    return( 0 ) ;
  }
  printf("Invalid add SAP command\n") ;
  print_help() ;
  return( 1 ) ;
}


/* "add" requested - sub-selection  */
int add_sub_select()
{
  char  add_cmd[10];
  int     retc=0;

  strcpy(add_cmd, strtok( cmd_string, " "  )) ;
#ifdef _DEBUG
  printf( "add_sub_select: add_cmd\t\t= %s\n", add_cmd );
#endif
  switch (cmd_to_table(add_cmd)) {
  case GTM_ADD_BOX:
      if (strcmp(current_graph, "Root") == 0) {
    printf("You cannot add a box graph to the Root submap\n") ;
    retc=-1;
      } else
    retc=add_box_graph();
    break ;
```

Figure 214 (Part 8 of 12). wtotapi1.c Source Code (Including Debug Information)

```
  case GTM_ADD_GRAPH:
#ifdef _DEBUG
  printf( "add_sub_select: current_graph\t= %s\n", current_graph );
#endif
      if (strcmp(current_graph, "Root") == 0)
    retc=add_root_graph() ;
      else
    retc=add_graph_graph() ;
    break ;

  case GTM_ADD_VERTEX:
      if (strcmp(current_graph, "Root") == 0) {
    printf("You must set the parent graph for vertex, use parent command\n");
    retc=-1;
      } else
    retc=add_vertex() ;
    break ;

  case GTM_ADD_ARC:
      retc=add_arc() ;
    break ;

  case GTM_ADD_SAP:
      retc=add_sap() ;
    break ;
  default:
    print_help() ;
    retc=-1;
  }
  return( retc );
}
/* Set the status of a vertex - up, down, marginal, unknown */
int set_vertex()
{
  int vert_status ;
  char   vertex_name[MAX_NAME_LEN] ;
  char   status_requested[8] ;

  sscanf(cmd_string, "%s %s", &vertex_name, &status_requested) ;
  if ( (vert_status = char_to_status(status_requested)) == EOF)
    return( -1 );
  printf ("status - %d\n", vert_status)  ;
  /* Just issue the request */

  nvotChangeVertexStatus( current_prot_nbr, &vertex_name, vert_status) ;
  printf("%s\n", nvotGetErrorMsg(nvotGetError())) ;
  return( 0 );
}

int parse( char cmd[10] )
{
  int      retc=0;

#ifdef _DEBUG
  printf( "parse: cmd\t\t= %s\n", cmd );
#endif

  switch (cmd_to_table(cmd)) {
    case GTM_SET_PROTOCOL:
#ifdef _DEBUG
      printf("parse: GTM_SET_PROTOCOL\n" );
      printf("parse: cmd_string\t= %s\n", cmd_string);
#endif
      retc=set_cur_protocol();
      break;
```

Figure 214 (Part 9 of 12). wtotapi1.c Source Code (Including Debug Information)

```
    case GTM_SET_PREFIX:
#ifdef _DEBUG
      printf("parse: GTM_SET_PREFIX\n" );
      printf("parse: cmd_string\t= %s\n", cmd_string);
#endif
      if( strcpy(oid_prefix, cmd_string))
        printf( "Sucessfull Operation. Prefix = %s\n", oid_prefix );
      break;
    case GTM_SET_PARENT:
#ifdef _DEBUG
      printf("parse: GTM_SET_PARENT\n" );
      printf("parse: cmd_string\t= %s\n", cmd_string);
#endif

      retc=set_cur_graph(); break;
    case GTM_ADD_OBJECT:
#ifdef _DEBUG
      printf("parse: GTM_ADD_OBJECT\n" );
      printf("parse: cmd_string\t= %s\n", cmd_string);
#endif
      retc=add_sub_select(); break;
    case GTM_SET_OBJECT:
#ifdef _DEBUG
      printf("parse: GTM_ADD_OBJECT\n" );
      printf("parse: cmd_string\t= %s\n", cmd_string);
#endif
      retc=set_vertex(); break;
    case HELP:
#ifdef _DEBUG
      printf( "parse: HELP\n" );
#endif
      retc=print_help();
      break;
    case QUIT:
#ifdef _DEBUG
      printf( "parse: QUIT\n" );
#endif
      retc=1;
      break;
    default:
#ifdef _DEBUG
      printf( "parse: default\n" );
#endif
      print_help();
      retc=-1;
   }
   return( retc );
}

int main(int argc, char **argv)
{
  int    quit = FALSE ;
  int    command_line = 0;
  int     opt ;
  int     rc=0;
  int      prompt = 1;
  int    com_line = 0;
  char   hostname[48] ;
  char   cmd[10] ;
  char   *proto = NULL ;
  char   *focus = NULL ;
  char   command[255] ;
  nvotReturnCode retc ;
  static unsigned int ovwdbTimeout = 5;

  strcpy(current_graph, "Root") ;
  gethostname(hostname, 48) ;  /*     talk to our own host */

  while((opt=getopt(argc, argv, "o:P:f:c:s:ph?")) != EOF)
    switch(opt) {
      case 'o':
```

Figure 214 (Part 10 of 12). wtotapi1.c Source Code (Including Debug Information)

```
#ifdef _DEBUG
    printf( "main: o selected\n" );
    printf( "main: oid prefix is %s\n", optarg );
#endif
    strcpy( oid_prefix, optarg);
        break;
      case 'P':
#ifdef _DEBUG
    printf( "main: P selected\n" );
    printf( "main: protocol is %s\n", optarg );
#endif
    proto = ( char * )malloc( sizeof( optarg ));
    strcpy( proto, optarg );
        break;
      case 'f':
#ifdef _DEBUG
    printf( "main: f selected\n" );
    printf( "main: focus is %s\n", optarg );
#endif
    focus = ( char * )malloc( sizeof( optarg ));
    strcpy( focus, optarg );
        break;
      case 'c':
#ifdef _DEBUG
    printf( "main: c selected\n" );
    printf( "main: command is %s\n", optarg );
#endif
    command_line = 1;
    strcpy( command, optarg);
        break;
      case 's':
#ifdef _DEBUG
    printf( "main: s selected\n" );
    printf( "main: server is %s\n", optarg );
#endif
    strcpy( hostname, optarg);
        break;

      case 'h':
#ifdef _DEBUG
    printf( "main: h selected\n" );
#endif
    print_help();
    exit( 0 );
    break;

      case 'p':
#ifdef _DEBUG
    printf( "main: p selected\n" );
#endif
    prompt=0;
    break;

      case '?':
#ifdef _DEBUG
    printf( "main: usage selected\n" );
#endif
        usage();
    exit( -1 );
        break;
   }

#ifdef _DEBUG
  if( strlen(command) > 1 ) {
  printf( "main: OID is\t\t%s\n", oid_prefix );
  printf( "main: Command is\t%s \n", command );
  }
#endif
```

Figure 214 (Part 11 of 12). wtotapi1.c Source Code (Including Debug Information)

```
  /* First, connect to gtmd */
  if(retc = nvotInit(hostname, FALSE, TRUE) != NVOT_SUCCESS) {
    printf("Open Topology connection failed\n") ;
    printf("%s\n", nvotGetErrorMsg(retc)) ;
    exit(-1) ;
  }
  if( prompt )
    printf("Open Topology connection to %s successful\n", hostname) ;
  /* Try for Synchronous operation */
  if (nvotSetSynchronousCreation(ovwdbTimeout) == NVOT_SUCCESS)
    if( prompt )
      printf("Open Topology calls will be synchronous\n") ;
  /* Set parameters specified on command line */

  if( proto ) {
#ifdef _DEBUG
    printf( "main: proto\t=%s\n", proto );
#endif
    strcpy( cmd_string, proto );
    set_cur_protocol();
  }

  if( focus ) {
#ifdef _DEBUG
    printf( "main: focus\t=%s\n", proto );
#endif
    strcpy( cmd_string, focus );
    set_cur_graph();
  }

  if( command_line ) {
#ifdef _DEBUG
    printf( "main: Command line mode\n" );
#endif
    strcpy( cmd, strtok( command, " " ));
    strcpy( cmd_string, strtok( NULL, '\n' ));
    rc=parse( cmd );
  } else {
#ifdef _DEBUG
    printf( "main: Interactive mode\n" );
#endif
    while ( quit == FALSE ){
      if(prompt)
        printf("Command> ") ;
      if( scanf( " %[¬\n]", command ) == EOF )
        quit=TRUE ;
      else {
#ifdef _DEBUG
        printf( "main: command = %s\n", command );
#endif
        strcpy( cmd, strtok( command, " " ));
        strcpy( cmd_string, strtok( NULL, '\0' ));
#ifdef _DEBUG
        printf( "main: cmd\t= %s\n", cmd );
        printf( "main: cmd_string\t= %s\n", cmd_string );
#endif
        if( parse( cmd ))
          quit=TRUE ;
      }
    }
  }
  /* Finished with gtm connection */
  nvotDone() ;
  return( rc );
}
```

Figure 214 (Part 12 of 12). wtotapi1.c Source Code (Including Debug Information)

Index

Printed in U.S.A.

SG24-4515-00