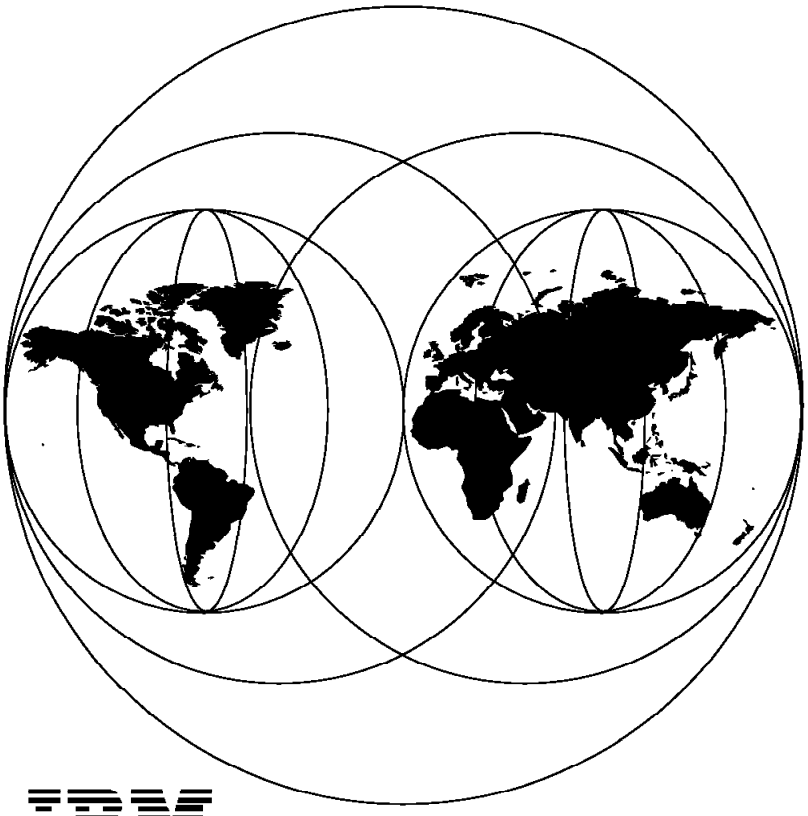
International Technical Support Organization

GG24-4327-00

Examples of Using NetView for AIX

November 1994



IBM

International Technical Support Organization
Raleigh Center

International Technical Support Organization

GG24-4327-00

Examples of Using NetView for AIX

November 1994

Take Note!

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

First Edition (November 1994)

This edition applies to Version 2 Release 1 and Version 3 of IBM NetView for AIX (previously known as IBM SystemView NetView/6000), Program Number 5696-731 for use with the RISC System/6000 AIX Version 3 Release 2, Program Number 5756-030.

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. 545, Building 657
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 provides an overview and examples of using NetView for AIX V3R1. It assumes the reader has a general knowledge of NetView for AIX V2, as well as a good understanding of AIX system and network management.

This document is only a summary of some initial uses of NetView for AIX V3R1 and is not intended to imply that all examples provided here are the extent of possibilities of involving NetView for AIX in an enterprise's network and systems management activities. The reader of this document should have access to development division documentation regarding NetView for AIX and the family of products, particularly as relates to Systems Monitor for AIX, AIX LMU/6000, LNM for AIX, and AIX SNA Manager/6000.

This document is intended for personnel who need information related to the marketing and acceptance of network management products in an AIX V3 environment. A general knowledge of overall IBM and customer computing systems and C language programming is assumed.

(300 pages)

Contents

Figures

Special Notices

This publication is intended to help network management professionals to become familiar with examples of using NetView for AIX, together with the NetView for AIX family of products. The information in this publication is not intended as the specification of any programming interfaces that are provided by NetView for AIX. See the PUBLICATIONS section of the IBM Programming Announcement for NetView for AIX for more information about what publications are considered to be product documentation.

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

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

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

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (VENDOR) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

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

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

ACF/VTAM
AIX
AIX/6000
AIXwindows
IBM
MVS/ESA

NetView	OS/2
VM/ESA	SQL
DB2	DB2/6000
RISC System/6000	S/390

The following terms are trademarks of other companies:

UNIX	X/Open Company, Ltd.
HP	Hewlett-Packard Company
OSF and OSF/Motif	Open Software Foundation, Inc.
DECmessageQ	Digital Equipment Corporation
Microsoft, Windows	Microsoft Corporation
Oracle	Oracle Corporation
Informix	Informix Software, Incorporated
Novell and NetWare	Novell, Incorporated
Sun, Solaris, NFS, Network File System	Sun Microsystems, Incorporated

Preface

This document provides an overview and examples of using NetView for AIX V3R1. It assumes the reader has a general knowledge of NetView for AIX V2, as well as a good understanding of AIX system and network management.

This document is only a summary of some initial uses of NetView for AIX V3R1 and is not intended to imply that all examples provided here are the extent of possibilities of involving NetView for AIX in an enterprise's network and systems management activities. The reader of this document should have access to development division documentation regarding NetView for AIX and the family of products, particularly as relates to Systems Monitor for AIX, AIX LMU/6000, LNM for AIX, and AIX SNA Manager/6000.

This document is intended for personnel who need information related to the marketing and acceptance of network management products in an AIX V3 environment. A general knowledge of overall IBM and customer computing systems and C language programming is assumed.

How This Document is Organized

The document is organized as follows:

- Chapter 1, "Overview of NetView for AIX V3R1"

  This chapter provides an introduction to and overview of NetView for AIX V3R1.

- Chapter 2, "Discovery"

  This chapter provides a discussion of discovery as used by NetView for AIX.

- Chapter 3, "Database Extensions"

  This chapter provides a summary of NetView for AIX V3R1 database support.

- Chapter 4, "Event Configuration"

  This chapter provides a summary of NetView for AIX event configuration and shows examples of using NetView for AIX V3R1 event configuration support.

- Chapter 5, "NetView for AIX Open Topology"

  This chapter provides a summary and examples of using NetView for AIX open topology support.

- Chapter 6, "Manager Takeover"

  This chapter provides a summary and examples of manager takeover support provided in NetView for AIX V3R1.

- Chapter 7, "wtdriver6/wteuiap6 Sample NetView for AIX EUI API"

  This chapter gives a brief introduction and overview of using wtdriver6/wteuiap6, a sample user-written implementation of the NetView for AIX end user interface API.

- The following appendixes are included:

  Appendix A, "Open Topology MIB Reference"
  Appendix B, "Automatic Seed File Example Programs"
  Appendix C, "Open Topology Program Samples"

Appendix D, “Database Samples”
Appendix E, “NetView for AIX Default Events”
Appendix F, “Nvevents X11 app-defaults File”
Appendix G, “Selected AIX SNA Server Profiles”

Portions of this document have been extracted from IBM documents related to the subject of AIX network management. Some such documents are referenced in the following ″Related Publications″ section below.

Related Publications

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

IBM NetView for AIX Concepts: A General Information Manual Version 2, GC31-6234

IBM NetView for AIX Programmer's Reference, SC31-6239

IBM NetView for AIX Programmer's Guide, SC31-6238

IBM NetView for AIX Application Interface Style Guide, SC31-6240

IBM NetView for AIX Installation and Configuration, SC31-6237

IBM NetView for AIX User's Guide for Beginners, SC31-6232

IBM NetView for AIX and the Host Connection, SC31-6235

IBM NetView for AIX Database Guide, SC31-7190

IBM NetView for AIX Problem Determination, SC31-6236

IBM NetView for AIX Administration Reference, SC31-8104

IBM NetView for AIX Administrator's Guide, SC31-7192

International Technical Support Organization Publications

IBM Systems Monitor Anatomy of a Smart Agent, GG24-4398 (Planned availability: December, 1994)

Examples of Selected Configuration and Customization Matters Involved With NetView for AIX and Its Family, GG24-2521 (Planned availability: December, 1994)

Examples of Using AIX NetView/6000 APIs, GG24-4059

A complete list of International Technical Support Organization publications, 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 technical bulletins (redbooks) online, VNET users may type:

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

How to Order ITSO Technical Bulletins (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 1-800-284-4721. Visa and Master Cards are accepted. Outside the USA, customers should contact their IBM branch office.

Customers may order hardcopy redbooks individually or in customized sets, called GBOFs, which relate to specific functions of interest. IBM employees and customers may also order redbooks in online format on CD-ROM collections, which contain the redbooks for multiple products.

Acknowledgments

The authors of this document are:

Pascal Batut, IBM France
Paul Fearn, IBM UK
Richard Hine, IBM UK
Lex Molenbroek, IBM Netherlands

The advisors for this project were:

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

Contributing, as well, to the effort in creating this document were:

Mike Chamberlain, IBM UK
Jose Pedro Pina Coelho, IBM Portugal
Ikuo (Gan) Iwamura, IBM Japan

Ernst Ziegler, IBM Germany
Valentina Nardecchia, IBM Italy
Yohichiroh Ishii, IBM Japan
Emma Locke, IBM UK

This publication is the result of a residency 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:

Martha Crisson
Virinder Batra
Marvin Boswell
Fred Niemi
Ken Chambers
Jim Chou
Judith Dietz
Richard Buckman
Tom Hemp
Jim Collins
and the entire NetView for AIX development group, IBM Raleigh

Roberto Bresil, Abel Gripp and Gerson Brizola
Product Development, IBM Brasil

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 Corporation 545/B657/BB110
Attn: Dave Shogren / Rob Macgregor
Building 657 Rm BB110
4912 Green Road
Raleigh NC 27604

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

Chapter 1. Overview of NetView for AIX V3R1

NetView for AIX Version 1 was first introduced during 1992. It was a solely a manager for TCP/IP networks, using the SNMP protocol. Many of the features of the first version have been carried forward, with enhancements, to Versions 2 and 3:

- Automatic IP network discovery
- Intuitive Motif-based graphical user interface
- Connectivity with S/390 NetView
- General-purpose SNMP applications, such as the MIB Browser, monitoring, thresholding and data collection applications.

NetView for AIX Version 2, introduced during 1993, expanded greatly on the Version 1 base. In summary the enhancements were:

- Application Programming Interfaces (APIs) introduced to enable other applications to use NetView for AIX as a platform product
- Support for non-IP network topologies, based on the IBM generic topology model
- Graphical user interface enhancements to improve navigation and window management
- Event display and filtering enhancements

With the advent of NetView for AIX V3R1 the product is further enhanced, in several areas:

- *Openness*. Additional relational database support provides better access to data collected by NetView for AIX.
- *Usability*. Enhancements to the event-handling system improve the users' effectiveness.
- *Function*. New capability added for manager fallback and IP discovery.

1.1.1 NetView for AIX V3R1 and HP OpenView V3.3

NetView for AIX uses code licensed from Hewlett-Packard Company for part of its function. The level of OpenView code incorporated into NetView for AIX V3R1 is V3.3, except in the cases where IBM-developed features already deliver an equivalent function.

1.2 Summary of Selected NetView for AIX V3R1 Enhancements

This section summarizes selected enhancements included in NetView for AIX V3R1. We will explore these in more detail in later chapters.

NetView for AIX V3R1 introduces changes in the following areas:

- Discovery

  NetView for AIX incorporates automatic discovery of IP network topology, based on information gleaned from SNMP agents. NetView for AIX V3R1 provides several enhancements to this:

- Support for multiple subnet masks. This has also been shipped as a PTF for Version 2.

  Previously, if different subnet masks were used within a network, the topology displayed might not accurately represent the real layout. For example, it is common practice to assign ′restrictive′ subnet masks (such as 255.255.255.252) to point-to-point connections in order to save IP network address ranges. The rest of the network would normally use a less restrictive subnet mask such as 255.255.255.0.

  Prior to the change, NetView for AIX would apply the less restrictive mask to the point-to-point links, making them all appear to be connected into one subnetwork. Following the change, NetView for AIX will accurately represent the network, and will show the links as direct connections, rather than interposing a network symbol.

- Support for more router interfaces.

  With previous releases NetView for AIX had a limit of 22 interfaces per router. With NetView for AIX V3R1 this has been increased to 2000.

- Database

  NetView for AIX stores, topology, status, fault, performance and other information about managed nodes in a series of databases and log files. NetView for AIX V3R1 introduces the use of relational databases to several of these, specifically:

  - The IP topology database
  - The event log
  - The performance data collection database

  The relational database topic is covered in detail in Chapter 3, “Database Extensions” on page 33.

- Event configuration

  When certain errors occur on the network an event will be sent to the NetView for AIX V3R1 management host. These events are displayed via the event display application. Remote SNMP agents will also raise events to inform NetView for AIX V3R1 of any incidents or errors.

  In NetView for AIX V3R1 a number of new features have been added to the Event configuration.

  - Enhanced User interface
  - Multiple Dynamic Event Display Screens
  - Larger Event Window displays

- Manager takeover (Backup manager)

  With this feature any NetView for AIX V3R1 can be a Backup manager for resources in an IP Internet network, originally managed by other NetView for AIX systems. You can design an IP Internet network so that each manager in your network has its own Sphere of Control (SOC).

  Manager function can be provided by either AIX NetView/6000 V2R1, or NetView for AIX V3R1, but Backup manager function, can only be provided by NetView for AIX V3R1. If so designed, a Backup manager takes over management functions of a Manager that for any reason goes down. As

soon as that manager is available again, the Backup manager is informed, so it can stop his management functions, taken over earlier.

The mechanism used for this feature is the node down / node up traps, sent by manager nodes.

- Interactions with Systems Monitor for AIX

  With Systems Monitor for AIX Version 2 the addition of Segment managers enhance the distributed management facility from within NetView for AIX V3R1 by taking some of the functions away from NetView for AIX V3R1.

Chapter 2. Discovery

This chapter provides a discussion of discovery as used by NetView for AIX.

2.1 What is Discovery in NetView for AIX?

The NetView for AIX discovery process is the mechanism that obtains the network-related information and feeds the configuration to the NetView for AIX map application.

2.2 Open Technology

There are a number of processed involved discovering non-IP devices within the network. The open technology discovery daemon is called noniptopod and is informed of the remote agents by netmon raising the relevant trap. This trap will reach the noniptopod daemon if the object discovered has an IP address or is an agent such as the LMU/2 agent for OS/2. This area is covered in the Open technology chapter in this book.

2.3 Discovery and Polling Daemons

Within NetView for AIX there are a number of discovery and polling processes. These are:

- Discover a new entity on the network.
- Discover new devices from a device already discovered. For example: When a new node is discovered, this node may contain more information relating to other devices on the network, for example additional IP devices that this particular node can communicate with.
- Poll nodes for any status changes that may have occurred.
- Poll nodes for any configuration changes.

2.3.1 The netmon Daemon

The process that deals with the discovering of network entities NetView for AIX is called the netmon daemon. When NetView for AIX is started for the first time the netmon daemon will send an SNMP trap directly to the trapd daemon specifying newly discovered entities. These entities may be devices or remote agent software (for example, non-IP communications). The only requirement is that the discovered entity has an IP address, (for example, 9.24.104.23).

The netmon daemon will then poll the SNMP agent for additional information. This information includes:

- Configuration
- Topology
- Status changes

The netmon daemon has no direct communication with non-IP entities on the network; such devices are monitored and polled via the noniptopod daemon.

Netmon will discover the following network objects:

- The local network segment.
- All nodes on the local segment.
- All Routers and gateways on the segment.
- All Segments or networks attached to the gateways and routers. Although netmon will discover these entities, it will inform NetView for AIX that they should be initially configured in an *UNMANAGED* state.

The following information is obtained by the netmon daemon and inserted into the NetView for AIX object database:

System Description (sysDesc)	Name and version of the system's hardware, software and operating system.
System Object ID (sysObjectId)	Specifies the object ID for the device. This ID is used for a identify the specific object on the network.
Forwarding Status	Indicates if the entity is acting as an IP gateway to forward datagrams received.
IP Address table (ipAddrTable)	List addressing relevant to this device.
Interface table	List interface details such as token-ring cards.
System Location	The physical location of the device.
System contact	The system contact for this device.

If the entity on the network has been discovered, then the process flow is from top to bottom. If any changes are made to the IP map by the network manager then the process flow is reversed.

Figure 1 on page 6 shows these processes.

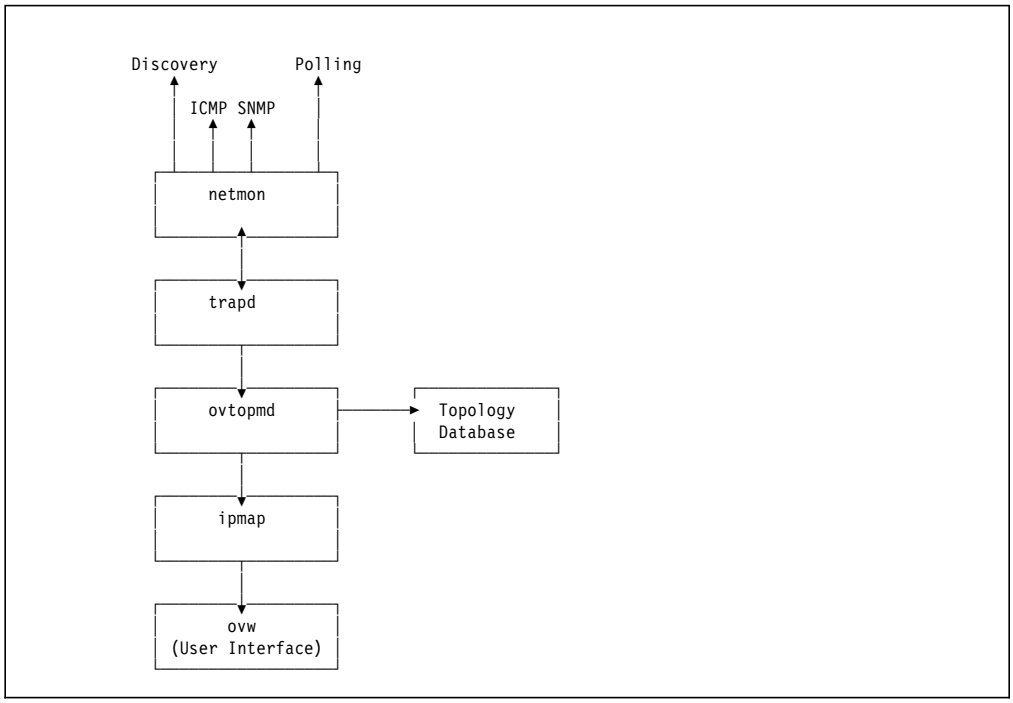


Figure 1. NetView for AIX V3 Discovery Daemons

The next sections explain the responsibilities of each of these daemons.

2.3.2 The trapd Daemon

The trapd daemon receives traps directly from the netmon daemon during initial discovery informing on new entities located in the network. Also netmon will raise a trap and forward it to the trapd daemon during polling to reveal any network status changes.

The trapd daemon will also receives traps from other internal processes and informs netmon.

trapd will log all received traps in the log file called /usr/OV/log/trapd.log.

2.3.3 The ovtopmd Daemon

This daemon maintains the network topology database. This database holds the netmon polling values and information relating to the network objects and their relationships. The ovtopmd daemon generates and updates the topology database using status information obtained by the netmon daemon.

2.3.4 The ipmap Application

The map of the entire IP network is known as the ipmap. One of the tasks of the ipmap application is to keep the consistency of the graphical interface information (what the user sees) with the information held in the topology database. For example if a new node has been discovered, ipmap will then inform the ovw application of the type of connection symbols and icons it needs to display.

This application will be notified of any new devices that netmon discovers. It will then look-up the SNMP object ID and display the relevant Icon, such as the workstation Icon for the RS/6000 processor.

2.4 Address Resolution

The Address Resolution Protocol (ARP) is used to locate additional entities from one discovered node. The discovered node may or may not hold information relating to other connected nodes. The netmon daemon will execute an SNMP get on the routing table MIB.

These commands may be useful in resolving routing information:

```
rnetstat -r <hostname>
```

The output of this command is similar to the following:

```
Destination          Gateway              Type       Interface
default              6611ral.itso.ral.ibm remote     tr0;
itso.ral.ibm.com     rs60002.itso.ral.ibm direct     tr0;
9.67.32              rs600010.itso.ral.ib remote     tr0;
9.67.37              rs600010.itso.ral.ib remote     tr0;
9.67.38              rs600010.itso.ral.ib remote     tr0;
9.67.38.10           lablnm.itso.ral.ibm. other      tr0;
9.67.38.89           lablnm.itso.ral.ibm. other      tr0;
9.67.40              rs600010.itso.ral.ib remote     tr0;
9.67.46              rs600010.itso.ral.ib remote     tr0;
9.67.46              rs60005.itso.ral.ibm remote     tr0;
```

```
9.67.46.10          lablnm.itso.ral.ibm. other      tr0;
9.67.46.11          6611ral.itso.ral.ibm other      tr0;
9.67.46.128         rs600010.itso.ral.ib remote     tr0;
9.67.46.139         lablnm.itso.ral.ibm. other      tr0;
9.67.46.140         lablnm.itso.ral.ibm. other      tr0;
9.67.46.141         lablnm.itso.ral.ibm. other      tr0;
9.67.46.152         lablnm.itso.ral.ibm. other      tr0;
9.67.46.170         lablnm.itso.ral.ibm. other      tr0;
9.67.46.175         lablnm.itso.ral.ibm. other      tr0;
9.67.46.180         lablnm.itso.ral.ibm. other      tr0;
9.67.46.187         lablnm.itso.ral.ibm. other      tr0;
9.67.46.188         lablnm.itso.ral.ibm. other      tr0;
127                 localhost.0.0.127.in direct     lo0
```

The route table MIB can be browsed using the NetView for AIX GUI:

- Select an object on a submap.
- Select **tools->MIB Browser** from the pull down menu.
- Click on **mgmt** followed by **Down Tree**.
- Click on **mib-2** then **Down Tree**.
- Click on **ip** then **Down Tree**.
- Click on **ipRouteTable** then **Down Tree**.
- Click on **ipRouteEntry** then **Down Tree**.
- Click on **ipRouteDest** then **Down Tree**.
- Click on **Start Query**.
- The table will look similar to Figure 2 on page 9.

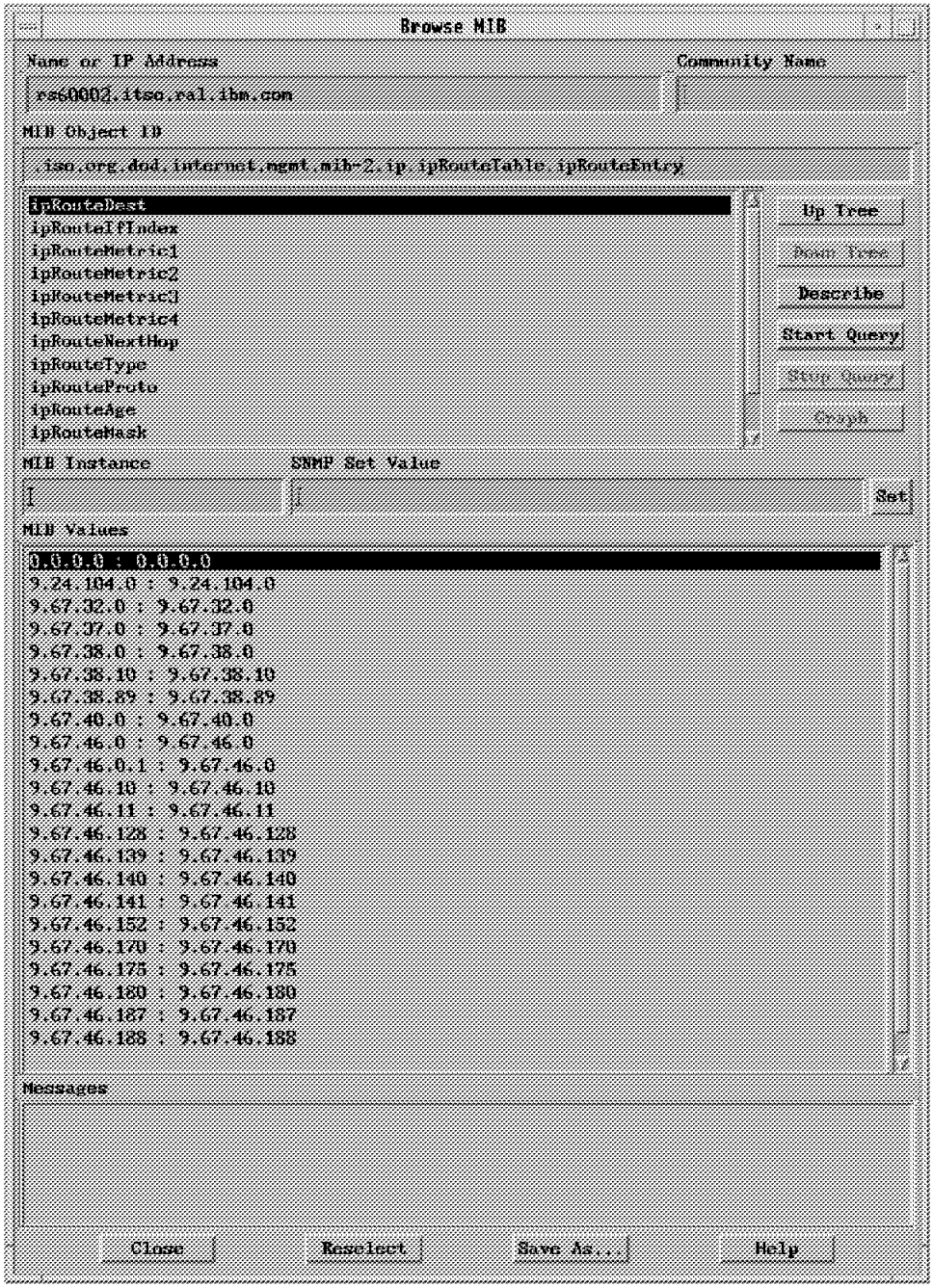


Figure 2. Browsing the Router Table MIB

2.4.1 Name Resolution

When an entity has been discovered the NetView for AIX process will attempt to resolve the IP address into a host name by:

- A *Domain Name Server* service such as DNS or NIS.
- Locating the address in the `/etc/hosts` file

The hostname will be used as the selection name for the entity on the NetView for AIX map.

For example, the node 9.24.104.28 has been discovered. NetView for AIX manages to resolve this IP address into rs60002.itso.ral.ibm.com. This value now becomes the selection name.

2.4.2 Selection Name and the Label

The purpose of the selection name is to make sure that every object has a textual name that can be displayed through the user interface. The selection name can be changed using NetView for AIX but this is not recommended. The label however is displayed on the map below the icon representing the object, this value is set initially to the selection name and can be modified by

- Select the object with the left hand mouse button.
- Select (using the right hand mouse button) **->Edit-> Modify/Describe->Symbol**
- Enter the new label name in the label field.
- Click on **Ok.**

  If the IP address cannot be resolved the selection name will be set to the IP address.

2.5 How to Configure the Discovery and Polling Options

This section describes the configuration options relating to the discovery and polling processes within NetView for AIX.

There are a number of configuration options for this process. It is important to get the configuration correct otherwise unnecessary network traffic will be generated.

2.5.1 SNMP Configuration

It is very important to set-up the SNMP configuration correctly before starting the discovery process. The objects will be discovered but if the SNMP community name is wrong then no additional information can be obtained from the SNMP agent running on that entity.

You may want to start NetView for AIX initially, then any incorrect requests for MIB information, (for example, invalid community names) will appear as events. SNMP may be updated and the NetView for AIX object database cleaned and NetView for AIX restarted.

The SNMP configuration for NetView for AIX can be viewed or modified using the interface. There are a number of reasons for changing the SNMP configuration. These are summarized below:

Specific Nodes This section contains specific nodes on the network for SNMP configuration.

IP Address wildcards This section contains IP addresses and will allow the use of wildcards for nodes that are to be SNMP managed.

The individual options are located in the bottom half of the screen. If these options are set incorrectly, then any valuable SNMP information cannot be obtained. A summary of the options:

Target The target address or hostname

Community The community name used for SNMP requests on the nodes specified in the *Target* option.

Set community The SNMP agent community name that the management agent will use to SET and MIB values. Agents are normally configured for different get and set community names.

timeout The timeout value for the SNMP Get or Set request. If there is no response from the agent, the NetView for AIX process will retry.

retry The retry value indicating the number of SNMP requests the process will attempt before failing.

Remote Port The UDP number on the target machine where the machine expects to receive SNMP requests. The standard port number is 161.

Status Polling This field indicated the frequency the netmon daemon will poll the nodes (using ping) to query the status. The more frequent the poll the more network traffic generated.

Figure 3 on page 11 shows a sample configuration for the SNMP agent for a RISC System/6000 machine.

Figure 3. SNMP Configuration

The example shows that the only specific node is the loopback address signifying the management station. For all the other nodes on the network the **Default** options will be used apart from the devices located on the network **9.24.104.***

If a node on a remote segment is consistently timing out you may want to include the segment as a separate option and increase the time-out value.

2.5.2 Topology/Status Polling Configuration

NetView for AIX discovery and polling status options and how to change these options are described in this section.

To see the polling options do the following from the main NetView for AIX screen:

- Select **Options->Topology/Status Polling Intervals: IP...**

This will show a screen similar to Figure 4.

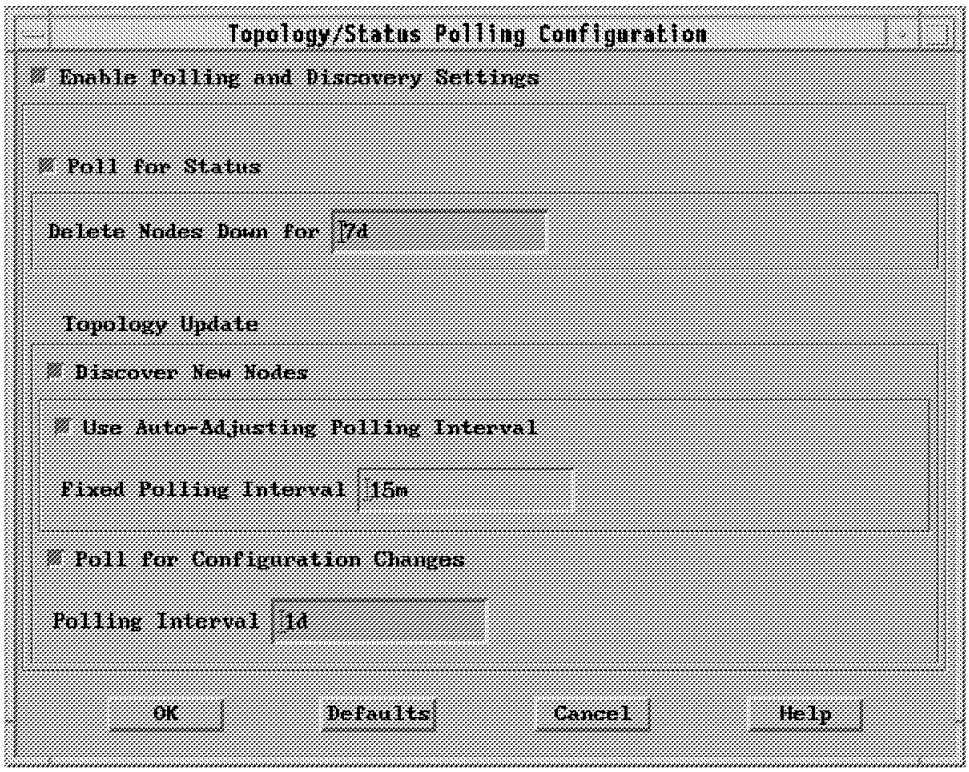


Figure 4. Topology/Status Polling Configuration

Figure 4 on page 12 shows the polling options. These values inform NetView for AIX when to activate the polling and discovery process. The options are as follows:

Enable Polling and Discovery Settings A global on/off switch for all the Polling and Discovery options.

Poll for Status NetView for AIX will poll nodes for status information. Nodes that do not respond within the time scale specified in the **Delete Nodes Down for** field will be **Deleted** from the map.

Discover New Nodes The NetView for AIX discovery process will poll existing SNMP nodes on the network to determine new nodes, for example the agents **ARP table** is examined.

Use Auto-Adjusting Polling Interval The NetView for AIX process initially polls the existing SNMP nodes at variable intervals, as fewer new nodes are found per polling cycle, the frequency of the polling interval will decrease. This allows less traffic as more nodes are discovered.

Fixed Polling Interval This option specifies the polling interval for discovery of new nodes.

Poll for configuration Changes The NetView for AIX configuration polling process will check for any actual configuration changes on the SNMP agent. This option will check the following information.

- Change in contact and location
- Forwarding IP packets
- Interface added
- Interface deleted
- Incorrect routing by a node
- Link address change
- Mismatch of link address
- Network mask change
- Node name change
- Object ID change
- Undetermined link address

Polling Interval The interval that netmon will use to check any configuration changes on each managed node.

2.5.3 The Seed File

One method of populating the NetView for AIX database is to use a seed file. This file contains a list of host names or IP addresses of SNMP agents within the network. Some reasons for using a seed file include:

- Limit the number of management stations. This option may be useful if you have only a certain number of critical machines in the network that you want managed. This option also requires the auto-discovery option to be switched off. See the example below.
- Speed up the initial discovery process. If the hostname or IP address are located in the seed file then the netmon daemon does not have to find this node but can get the configuration directly thus speeding up the process.
- A number of managed machines are outside the initial domain. NetView for AIX will only discover devices up to and including the first routers; devices beyond this point will have to be discovered manually or entered in the seed file.

The format of a seed file is a list of hostnames, IP addresses or IP address ranges within the management network with each device on a separate line. Routers beyond the first domain are useful additions to the seed file. This will allow NetView for AIX to discover the devices located on the other segments. As long as the router has ICMP traffic option *enabled*.

The following example shows how, with the use of a seed file, the discovery process is changed. The examples describe how to:

- Clear the NetView for AIX databases and restart the discovery with no seed file.
- Use the existing map and add the routing devices to the database using a seed file.
- Use a seed file to create a new map showing only selected router devices.

2.5.3.1 Initial Map Generation

The NetView for AIX database must be cleared to show this example. To do this:

- From the command line type `smit nv6000`
- Select **Maintain**.
- Select **Clear databases**.
- Select **Clear topology database**.

  Note: This will kill all user interface windows, clear all topology databases plus restore default IP discovery and polling defaults.
- Select **Done**.

The NetView for AIX databases are now cleared, ready for the discovery process to begin. Another way to clear the databases is via the following ″Restart automatic map generation″ smit option.

To restart the map generation process:

- Select **Control** from the smit nv6000 screen
- Select **Restart automatic map generation**

  Note: This will kill all user interface windows, clear all topology databases, the trapd.log, ovevent.log and ovevent.log.BAK and restore default IP discovery and polling defaults.

  Figure 5 on page 15 is the smit output from this action.

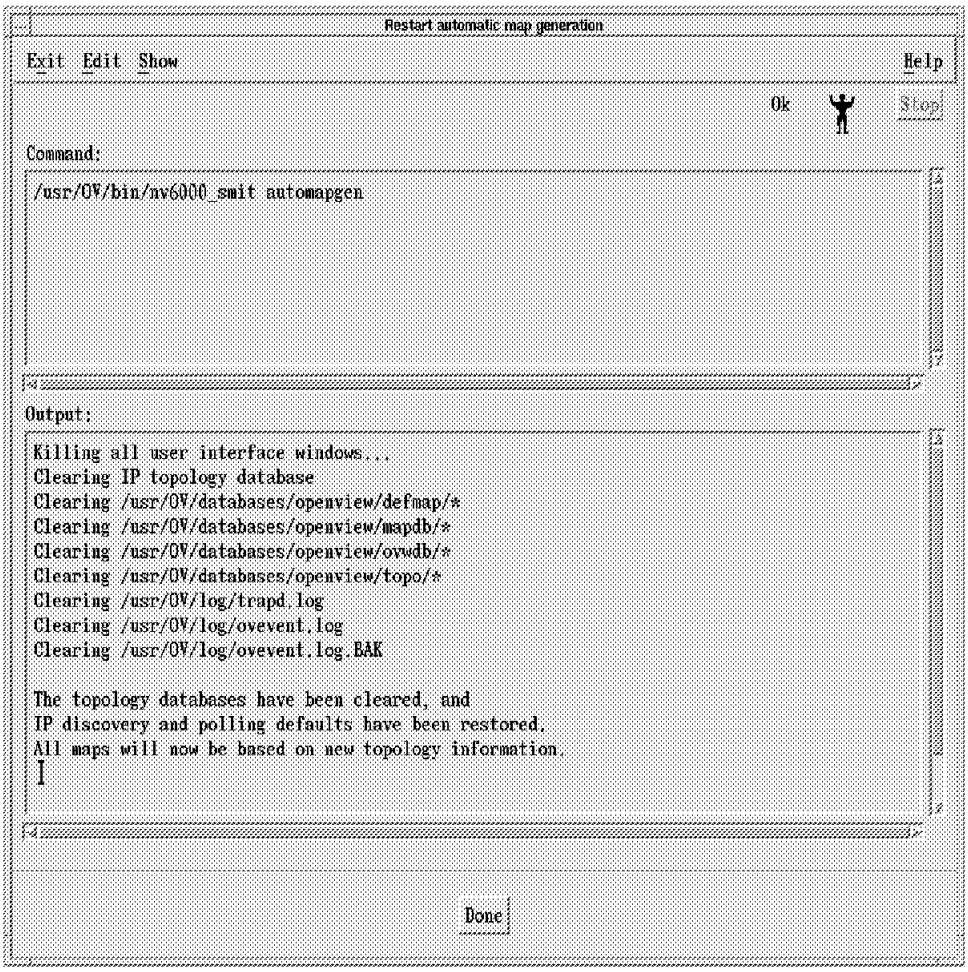


Figure 5. SMIT Panel After Restart Automatic Map Generation

- Exit from smit.
- Start the NetView for AIX End User Interface (EUI) and other daemons via smit or via a command such as:

  nv6000 -m somemapname

  In the following example, the following command was used, with a mapname of ITSO1:

  nv6000 -m ITSO1

  The initial discovery process has only found devices in the current segment. (See Figure 6 on page 16).

Notice that the map shows the local segment and all the router and gateway devices. The lighter colored icons signify devices attached to the routers and gateways; these are initially unmanaged.

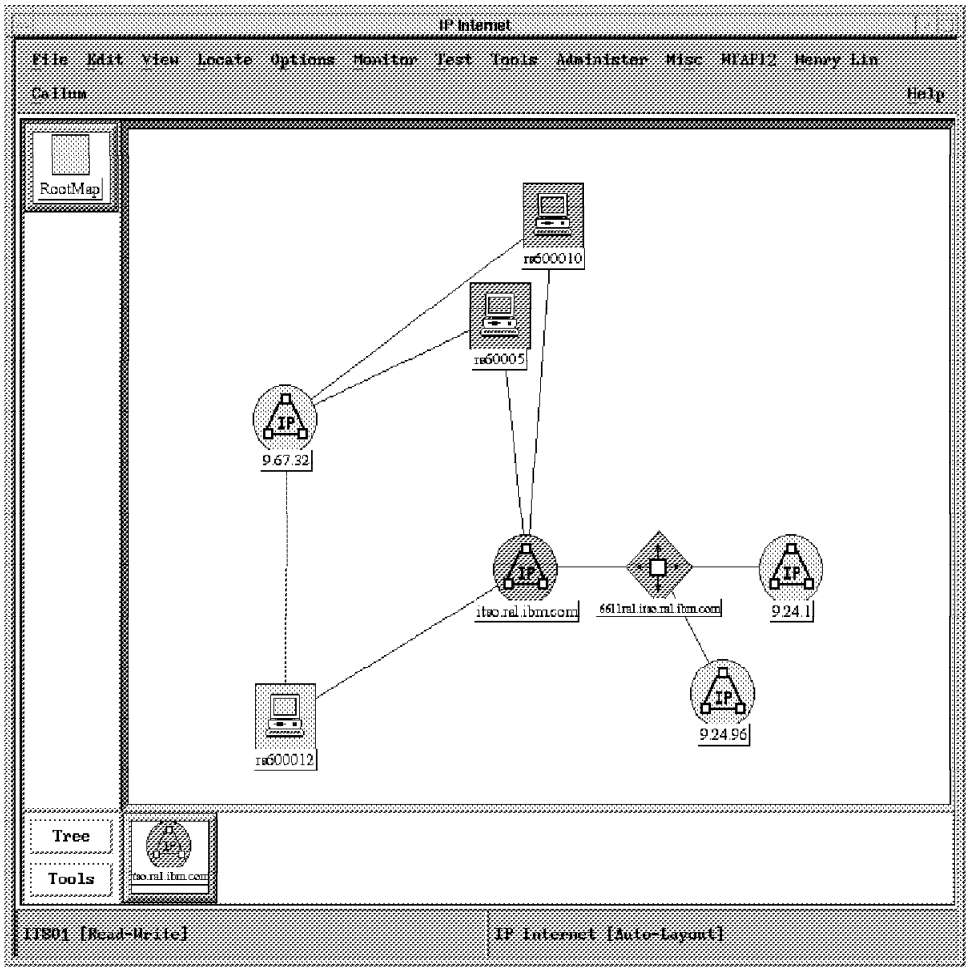


Figure 6. The NetView for AIX Map on Initial Startup

Figure 7 on page 17 shows the devices discovered in the local network. We will see later how this changes when we restrict discovery via a seed file in conjunction with automatic discovery being disabled.

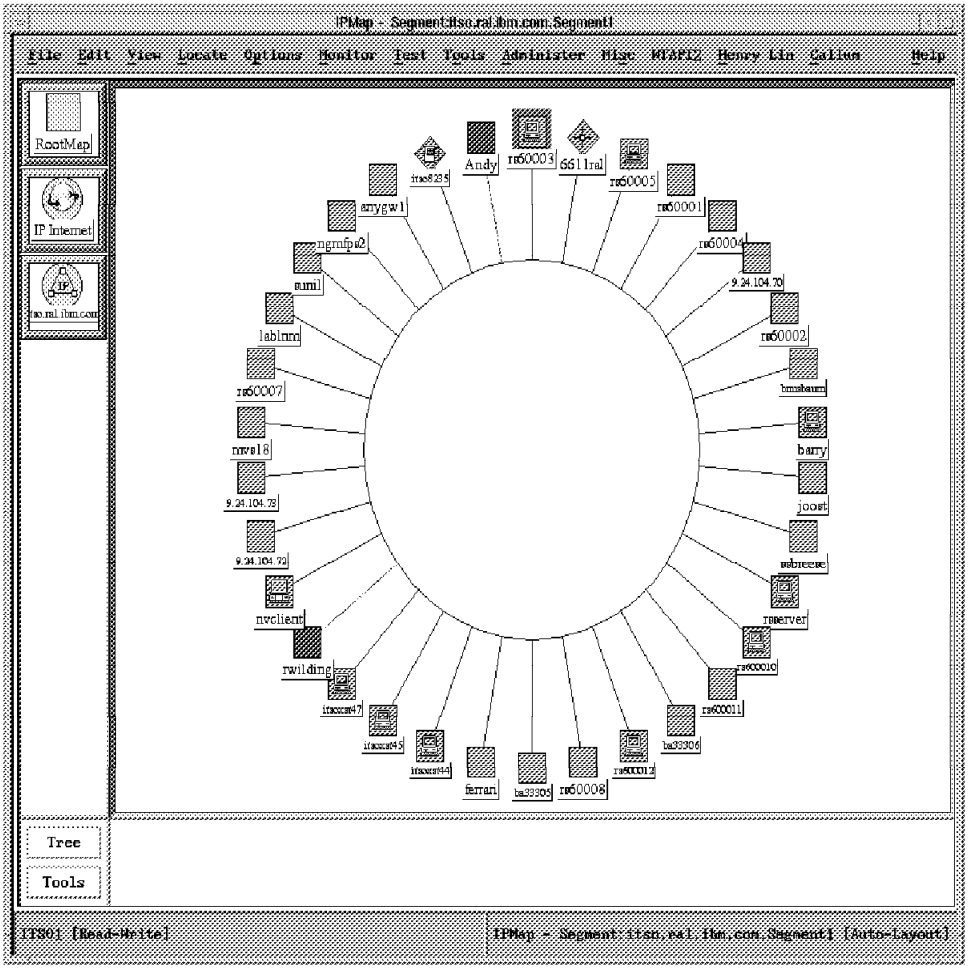


Figure 7. Local Segment with No Seedfile and Automatic Discovery On

2.5.3.2 Discovery of Router's IP Addresses in Seed File

Now we will include two router devices in a seed file and inform netmon that they will require network management from NetView for AIX.

- Using an editor (vi) create a seed file with the following IP addresses for the known routers on the network. The file created is called /u/paul/router_seed:

```
9.24.1.2
9.19.143.10
```

We would like to create a new database using the seed file, so clear the topology database as previously discussed in 2.5.3.1, "Initial Map Generation" on page 14.

To restart netmon using the new seed file:

- From the AIX command line type `smit nv6000`.
- Select **Configure**.
- Select **Set options for daemons**.
- Select **Set options for topology, discovery and database...**
- Select **Set options for netmon daemon**.

- Enter /u/paul/router_seed in the 'Load seed file from' field.
- Select **Do** followed by **Done**.

Then, the following command was used to restart the NetView for AIX EUI from the command line:

```
nv6000 -m ITSO1
```

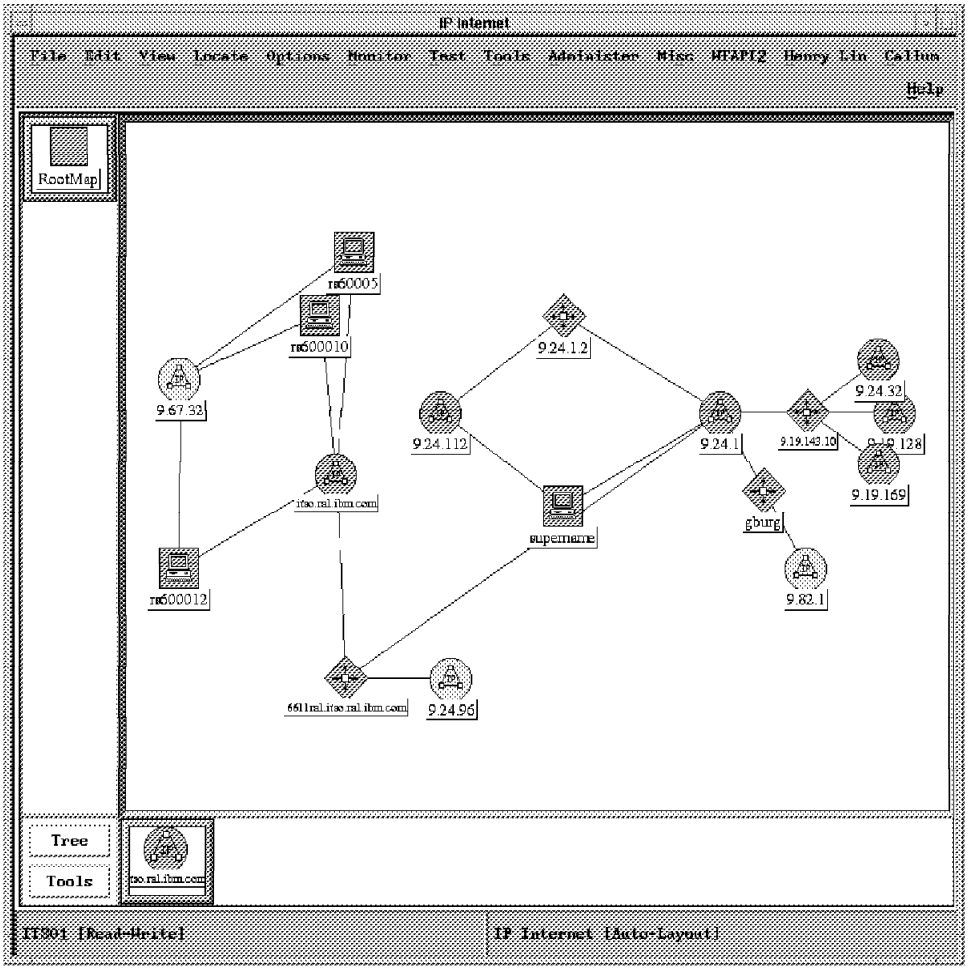


Figure 8. The NetView for AIX Map After the Seed File

By adding these routers to the seed file a number of new devices have appeared on the map. (See Figure 8). The light colored icons signify an unmanaged device. You can see that the new routers have been discovered and any devices discovered in the path to the new router are also represented as icons.

Figure 9 on page 19 again shows the devices discovered in the local network, since the router We will see later how this changes when we restrict discovery via a seed file in conjunction with automatic discovery being disabled.

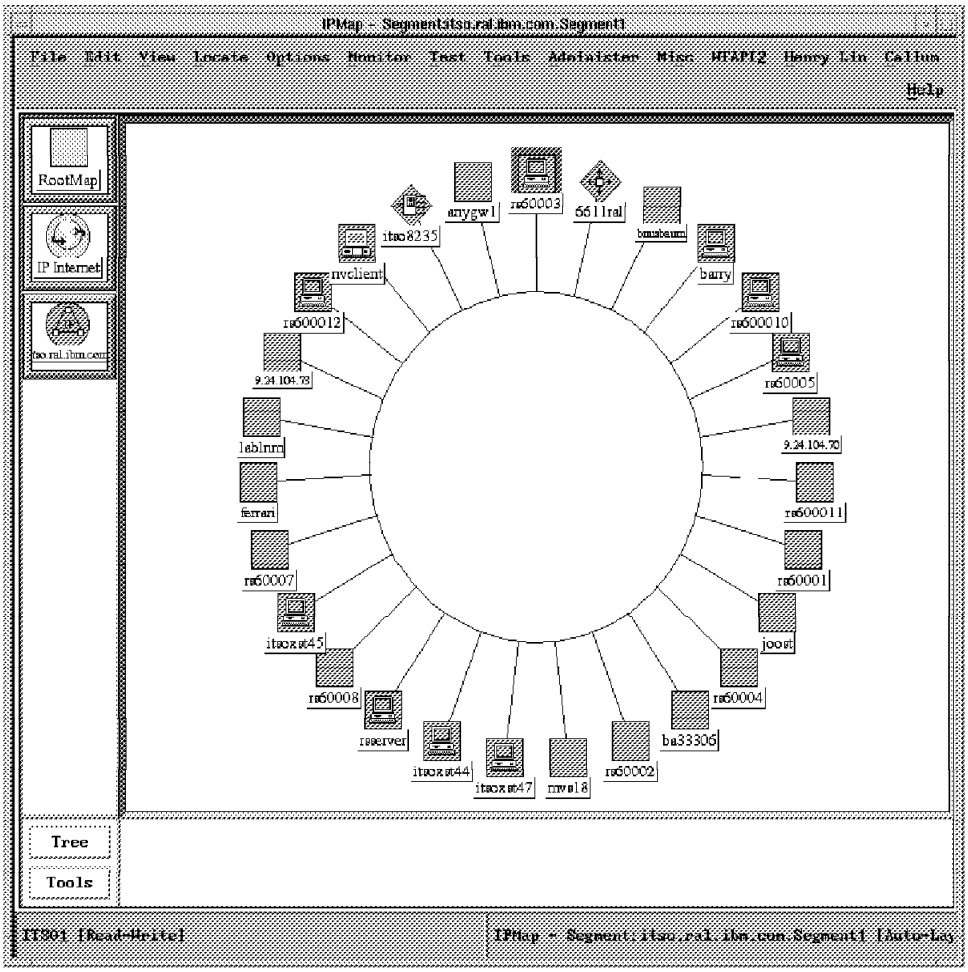


Figure 9. Local Segment with Seedfile and Automatic Discovery On

2.5.3.3 Creating a Map with Only the Router Devices in the Seed File

The final example shows how to inform netmon that it should only discover and inform IPMAP of the router devices on the network. The process is explained below:

Once again, we would like to create a new database using the seed file, so clear the topology database as previously discussed in 2.5.3.1, “Initial Map Generation” on page 14.

To turn *OFF* the Auto-discovery process:

- Using smit/control, stop the auto-discovery daemon (if not already stopped): netmon.
- Use the command:

  `nmpolling`

- Turn the Discover-New-Nodes to off as shown in Figure 10 on page 20.

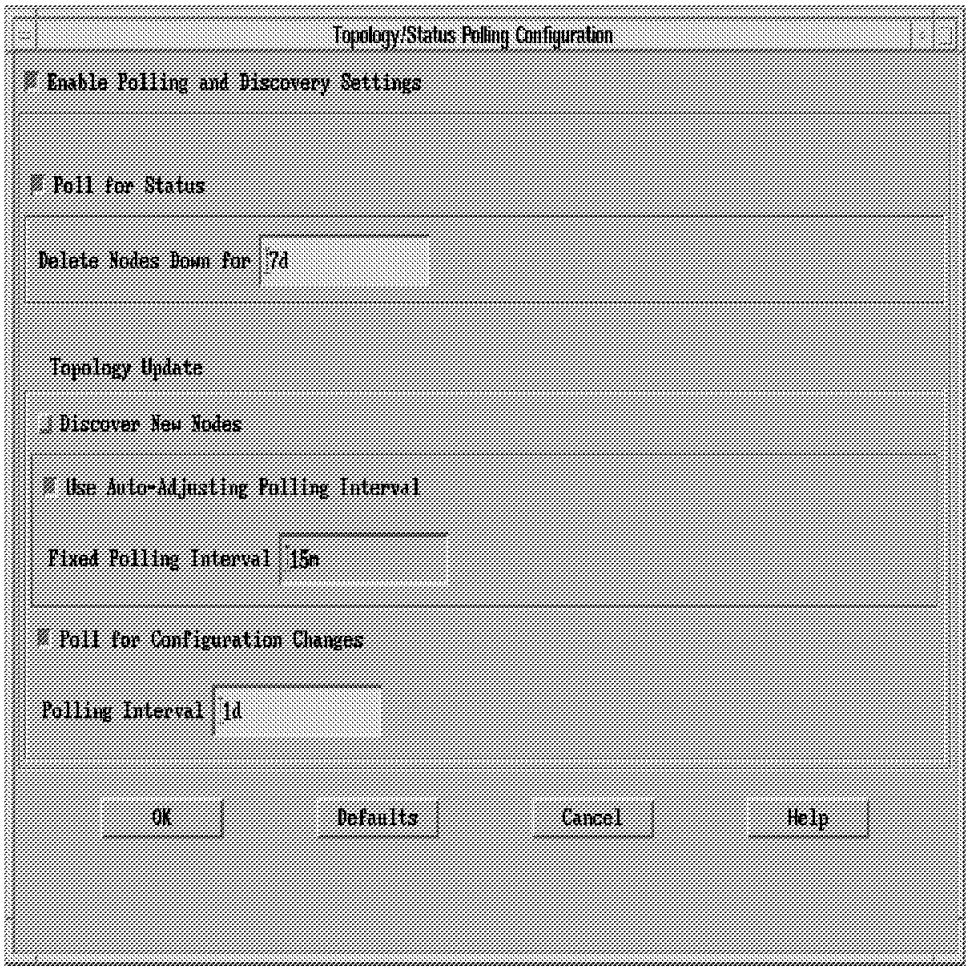


Figure 10. Turning Off Discover New Nodes. Note: this will done system-wide, for all operators/maps.

Use an editor to create a seed file containing the router devices. The file is called router_only.

```
9.24.1.2
9.19.143.10
9.24.1.4
9.19.141.241
9.19.129.202
9.19.129.132
```

To Restart netmon using the created seed file do the following:

- From the AIX command line type smit nv6000.
- Select **Configure**.
- Select **Set options for daemons**.
- Select **Set options for topology, discovery and database..**.
- Select **Set options for netmon daemon**.
- Enter /u/paul/router_only in the 'Load seed file from' field..
- Select **Do** followed by **Done**
- Quit from Smit.

- Type in `nv6000 -m somemapname` or use smit/control to bring up your mapname.

  The following command was used to restart the NetView for AIX EUI from the command line:

  ```
  nv6000 -m ITSO1
  ```

The new seed file will be processed and the routers are displayed on the map.

The screen looks like Figure 11. The map shows the router devices and objects connected to these routers. Again the light colored icons signify unmanaged nodes.

If an object is the seed file cannot be located in the network a message will be in the `/usr/OV/log/netmon.trace` file.

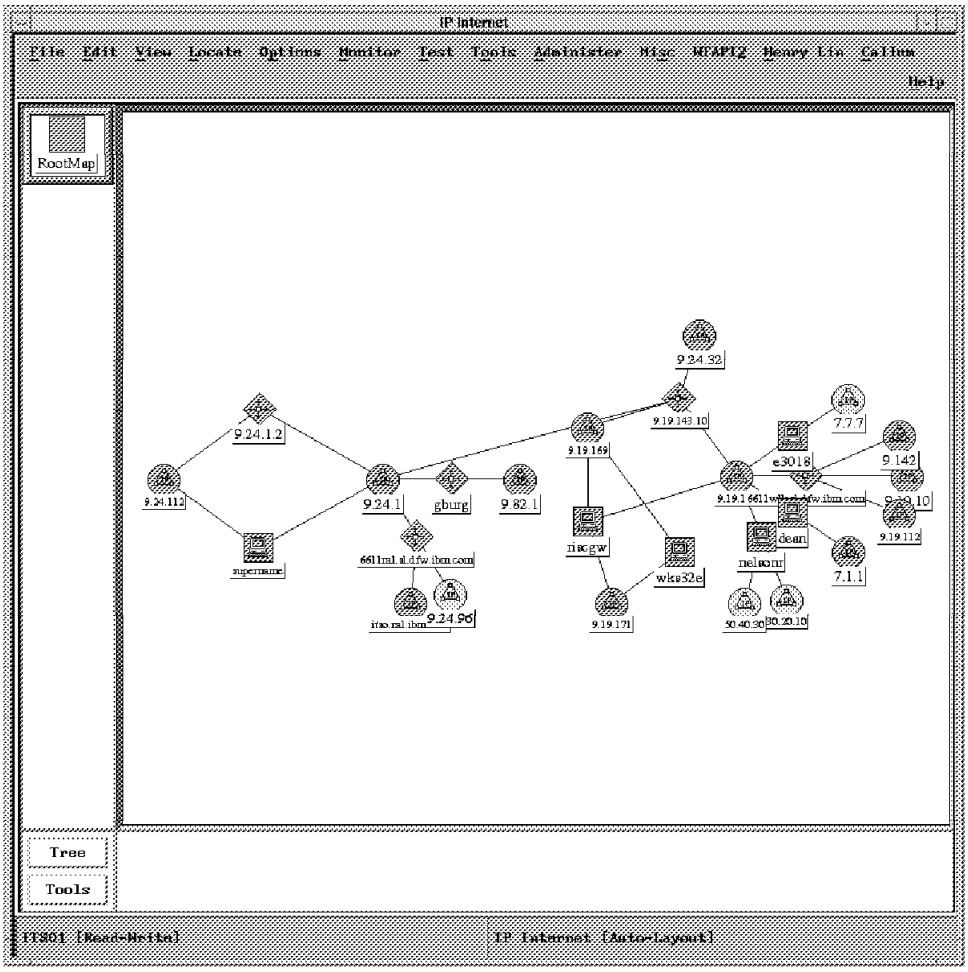


Figure 11. Auto Discovery Turned Off and Using Seed File

Figure 12 on page 22 shows the devices discovered in the local network; only the router and the manager's host (rs60003) is discovered since when we restricted discovery via a seed file and disabled automatic discovery.

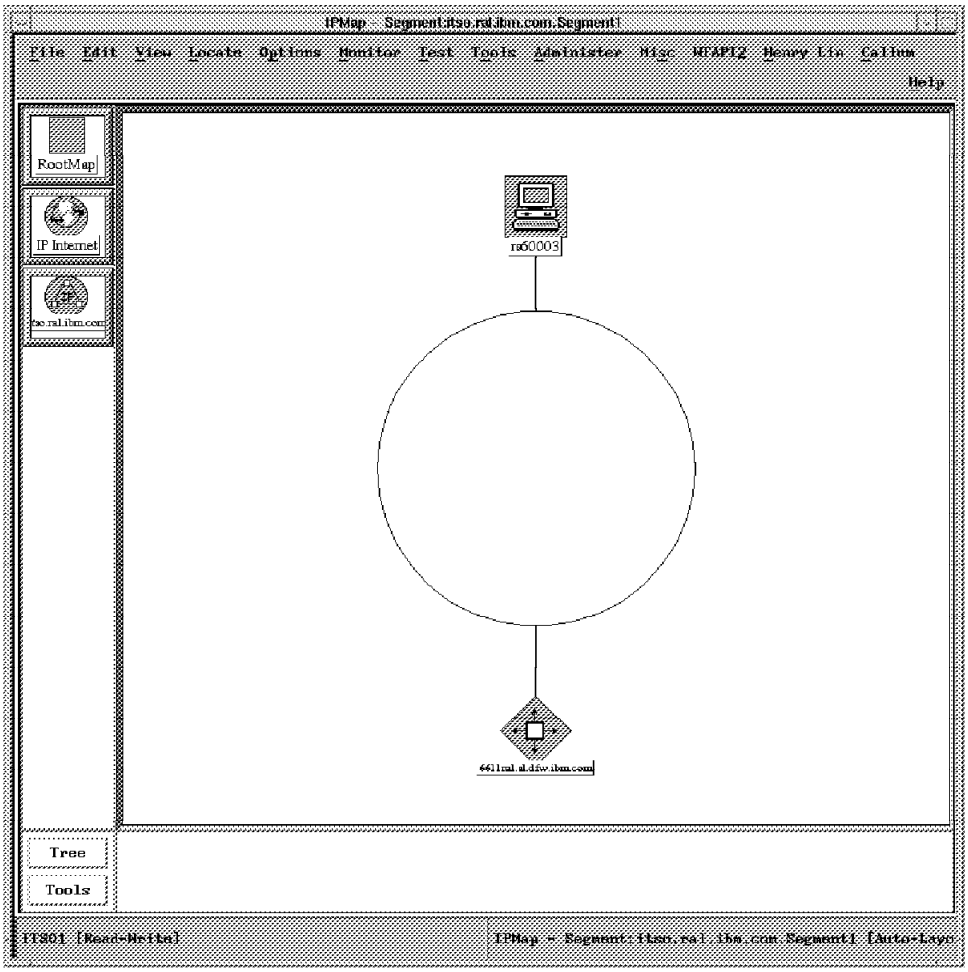


Figure 12. Local Segment with Seedfile and Automatic Discovery Off

2.6 More About the Discovery Process

The only objects discovered by the netmon process initially will be as follows:

- IP Networks, Gateways, and Routers on the Internet Map.
- Segments, Gateways, Routers, Hubs, and Bridges on the Network submaps.
- Hosts, Gateways, Routers, Hubs, and Bridges on the Segment Maps.

The relevant connections are be made by NetView for AIX for all the objects discovered. If any of the objects such as the hosts appear as incorrect objects on the map then the predefined value for that object may be incorrectly defined in the file `oid_to_type` file, and/or the `oid_to_sym` file.

The header and man pages for these files can be found on the NetView for AIX management system and in the *NetView for AIX Programmers Reference Guide,* SC31-6239. The following section summarizes the contents of these files:

2.6.1.1 The oid_to_type File

This file, found in /usr/OV/conf, is used by netmon to determine the correct IP topology attributes for the discovered object. Each Object discovered will have an Object ID associated with it, as one of its MIB values. To check a valid object ID for a specific device do the following: (6611ral is a valid ID in this example's IP network)

```
snmpget -c public 6611ral .1.3.6.1.2.1.1.2.0

The output shows the object Id as:
&.iso.org.dod.internet.private.enterprises.ibm.ibmprod.ibm6611

 which is resolved in dotted decimal form to:

 .1.3.6.1.4.1.2.6.2
```

The MIB browser could be used instead of the above snmpget.

The format of /usr/OV/conf/oid_to_type consists of four fields separated by colons. The fields contain:

- SNMP object Id
- The vendor (IBM, Sun, DEC)
- The agent type, such as IBM RS/6000 or HP9000/8000
- A set of flags controlling the topology attributes. Some of the flags are shown below:
  - G - Gateway(Router) device
  - B - Bridge or simple repeater
  - H - Multi-port repeater or Hub
  - I - Treat the device as if no SNMP support is available.
  - T - Terminal Server device.
  - U - Create the device in an UNMANAGED state.

As seen in the file /usr/OV/conf/oid_to_type, the 1.3.6.1.4.1.2.6.2 value has the options:

```
IBM:IBM 6611:G
```

This shows that when this type of device is discovered the discovery process knows that it is an IBM 6611 router.

The Object Id is also used to resolve the object ID into a symbol to represent this device on the Map. This file is called oid_to_sym and it is found in /usr/OV/conf/C. This file contains three fields:

- Object ID
- Class Member
- The Class member Name

The IBM 6611 router object ID is shows as follows:

```
1.3.6.1.4.1.2.6.2:Connector:Gateway
# IBM 6611 Router Processor
```

This shows that the Icon that will appear on the map will be a gateway of type connector. If we look at the Legend by selecting from the NetView for AIX main EUI menu **Help..Legend** the relevant icon that the IBM 6611 refers to is displayed.

If a different icon is required, then this file can be modified. To add a new Icon to the legend see the *NetView for AIX Programmers Guide,* SC31-6238.

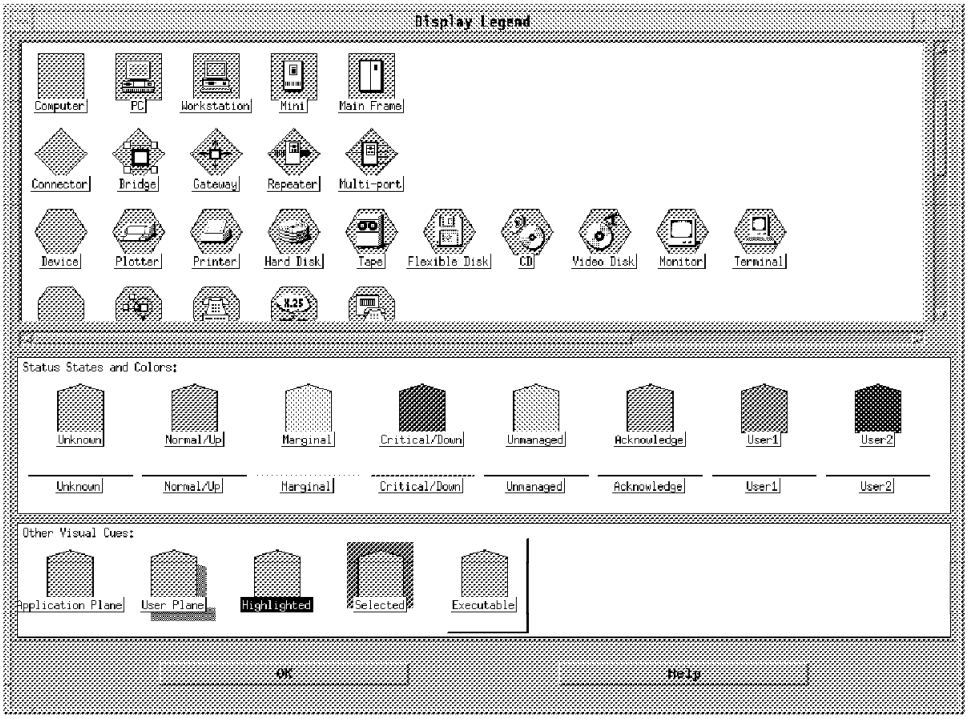


Figure 13. The Legend

2.6.2 Discovery Agent

Netmon uses the ICMP protocol to maintain the status of the managed nodes. If the NODE does not support the SNMP protocol then netmon will gain information using ICMP network Mask requests. See the file /usr/OV/conf/ovsnmp.conf

The options for running netmon are as follows:

-d Disable system checking. Netmon invokes a script called /usr/OV/bin/nmcheckconf to confirm the management system is correctly configured.

-J Causes netmon to attempt to speed up the process of discovering new nodes at the expense of limited broadcast traffic. When a new Node or network is discovered then Netmon causes the first capable node to broadcast ICMP echo requests. Thereafter while the node remains on the map netmon causes no additional broadcasts to be generated there.

This option should only be used where very few nodes support SNMP or where the use of proxy ARP is prevalent.

-m Sets the initial trace mask to tracemask. The default is No.

-s Refer to 2.5.3, “The Seed File” on page 13.

-S Discover a secondary address for devices, such as routers or gateways, that will support secondary addressing.

-u Supports additional discovery and management of IP nodes that exist in an IP network and are connected to the network that does not have an explicit IP address.

The trace mask values are:

- 0 Turn tracing off
- 1 Trace ICMP echo requests
- 2 ICMP echo requests and timeoutes
- 4 SNMP requests
- 8 SNMP requests and timeouts
- 16 traps generated
- 32 traps received

For example, when using netmon -M 2 The output from this command shows:

```
8079:sending Trap to 9.24.104.55 op=AT req=ARP
Output from the command for the 16
Traps raised:
UP Event: 9.24.104.123 (rs600011.itso.ral.ibm.com)
```

The netmon performance checks poll SNMP agents for the following configuration. The config file is found in /usr/OV/conf/ovsnmp.conf.

Changes are reported for the following:

- Arp Tables
- Broadcasts
- SNMP Requests
- What type of device will be discovered
- Why devices may not be discovered
- TCP/IP communication on network is necessary for discovery

2.7 Problem Determination

This section lists some commands that are useful in determining why some network objects may not being recognized by NetView for AIX

`ping hostname` Sends ICMP packet to the hostname. If the node is not already discovered by netmon, a ping to the node will start the process before the next netmon poll.

`snmpwalk hostname` Examines the MIB tree by stepping through each branch in turn.

`traceroute` Prints the route that the IP packets will take to a network host. (Note: This is a command not distributed with a particular product. It is available via various user disks).

Shared IP address If netmon discovers two or more nodes with the same IP address then it will begin to display and remove the Icon. If this is the case it is wise to switch on the -S option on netmon which will allow shared IP addresses. Note: This will increase network traffic.

`netmon -M` To switch on/off the tracing facility with netmon, this is a very useful debugging tool.

If some nodes appear on the map having strange IP addresses and do not appear to be connected with the main IP network, this may be due to an incorrect subnet mask setting. NetView for AIX can only draw networks correctly if the following objects have only one subnet mask per class.

- Gateways and routers
- The manager system
- Any nodes located in a seed file

Refer to NetView for AIX online help, online documentation and other standard approaches for assistance in problem determination.

2.7.1 Example of Online Help

The following figures in this section show using the online Help support of NetView for AIX to investigate "discovery".

NetView for AIX help is accessed via the main EUI pull-down, clicking on **NetView for AIX Library**.

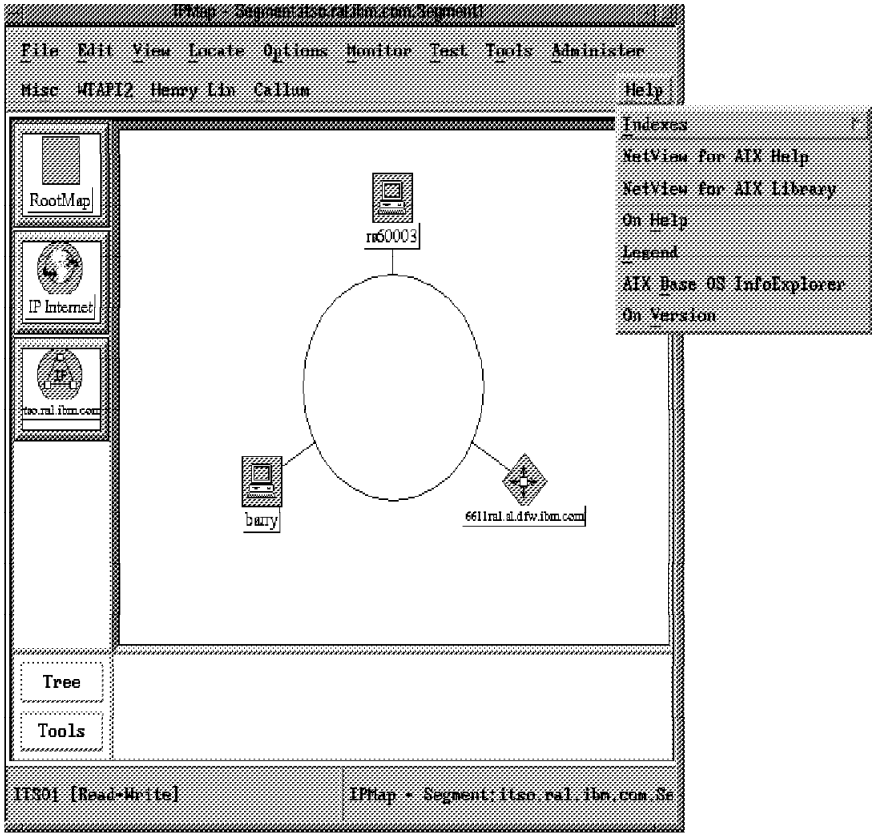


Figure 14. Accessing NetView for AIX Help. Heading toward NetView for AIX Library, in this example.

In this example, we wished to investigate ″discovery″ as found in the NetView for AIX Administrator's Reference.

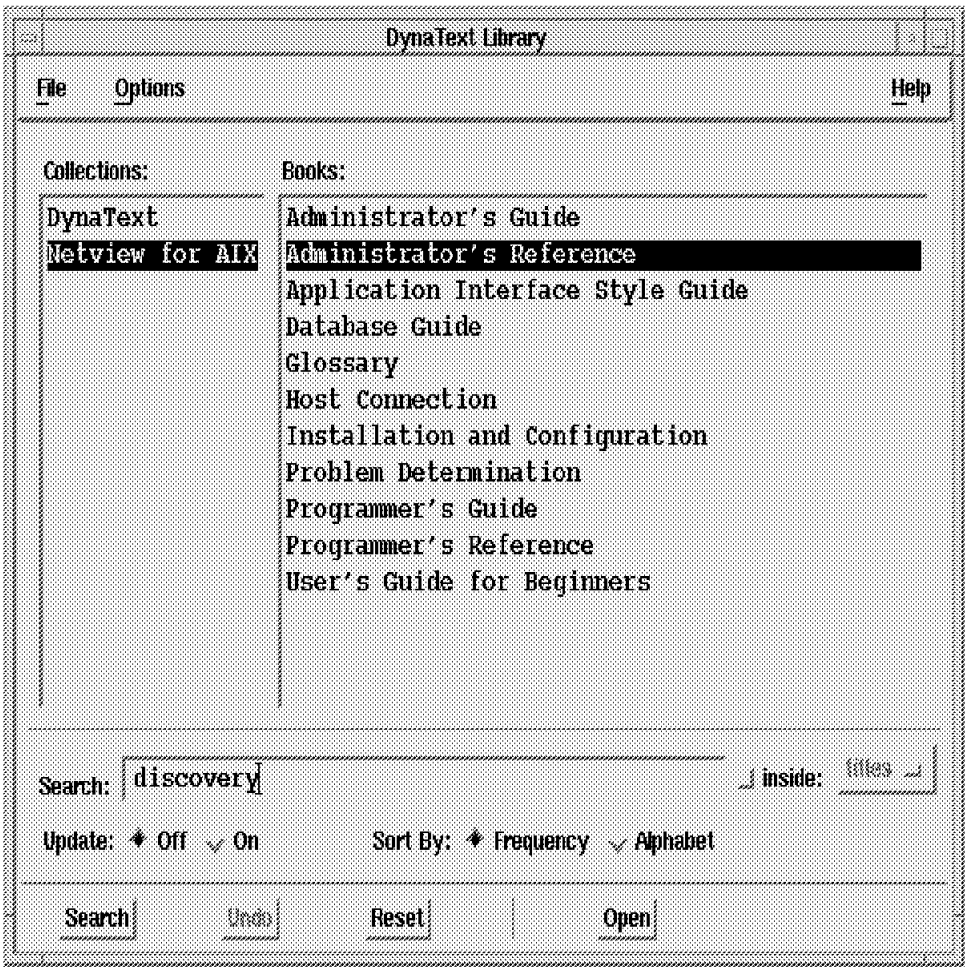


Figure 15. Selecting NetView for AIX and Administrator's Reference

Filling in the panel as indicated above and clicking on **Search** results in the following figure.

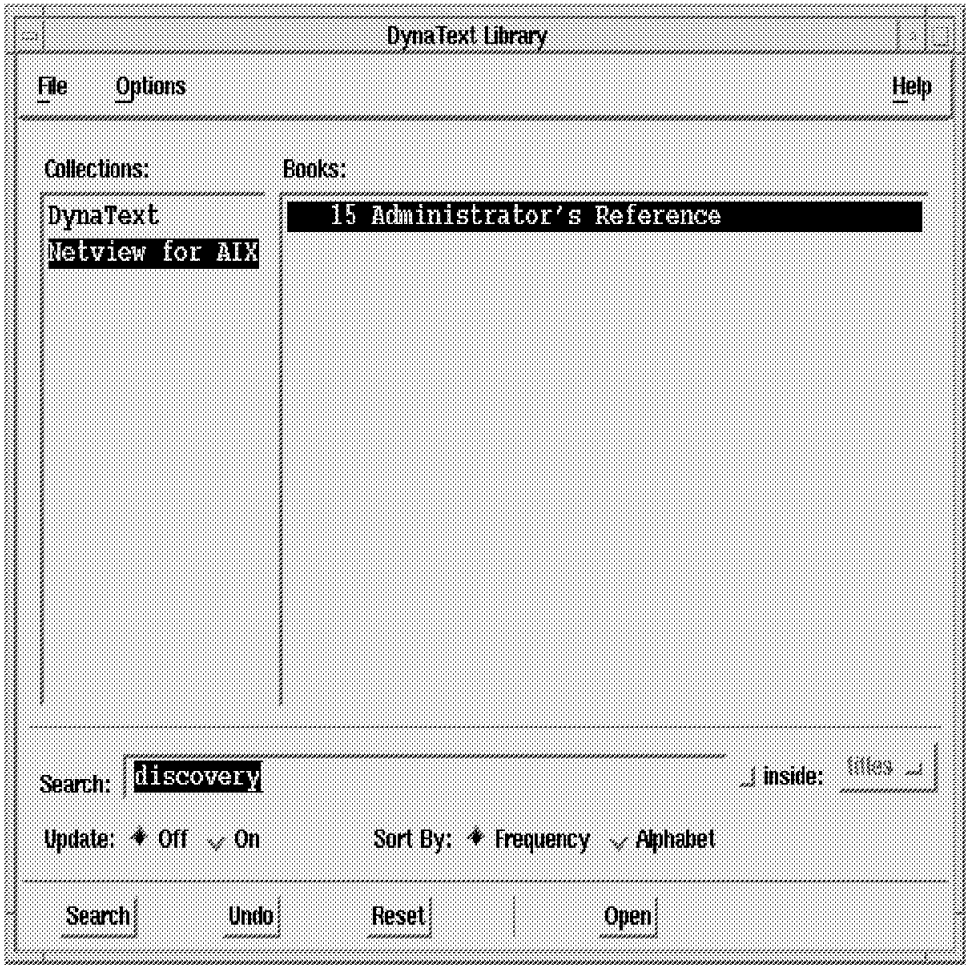


Figure 16. Showing 15 Found Occurrences

Selecting the 15 Administrator's References and clicking on **Open** results in the following figure.

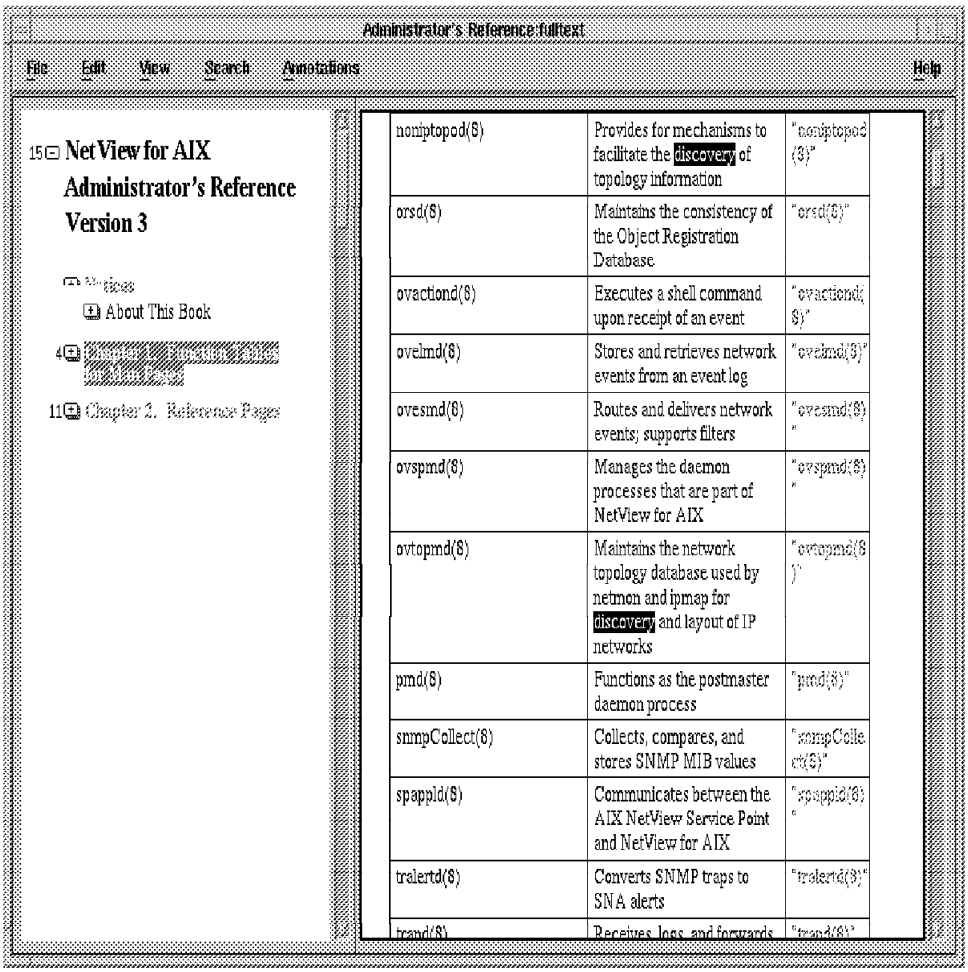


Figure 17. Panel for Full Text of Selected Document

Clicking on **Search** brings forward the following panel.

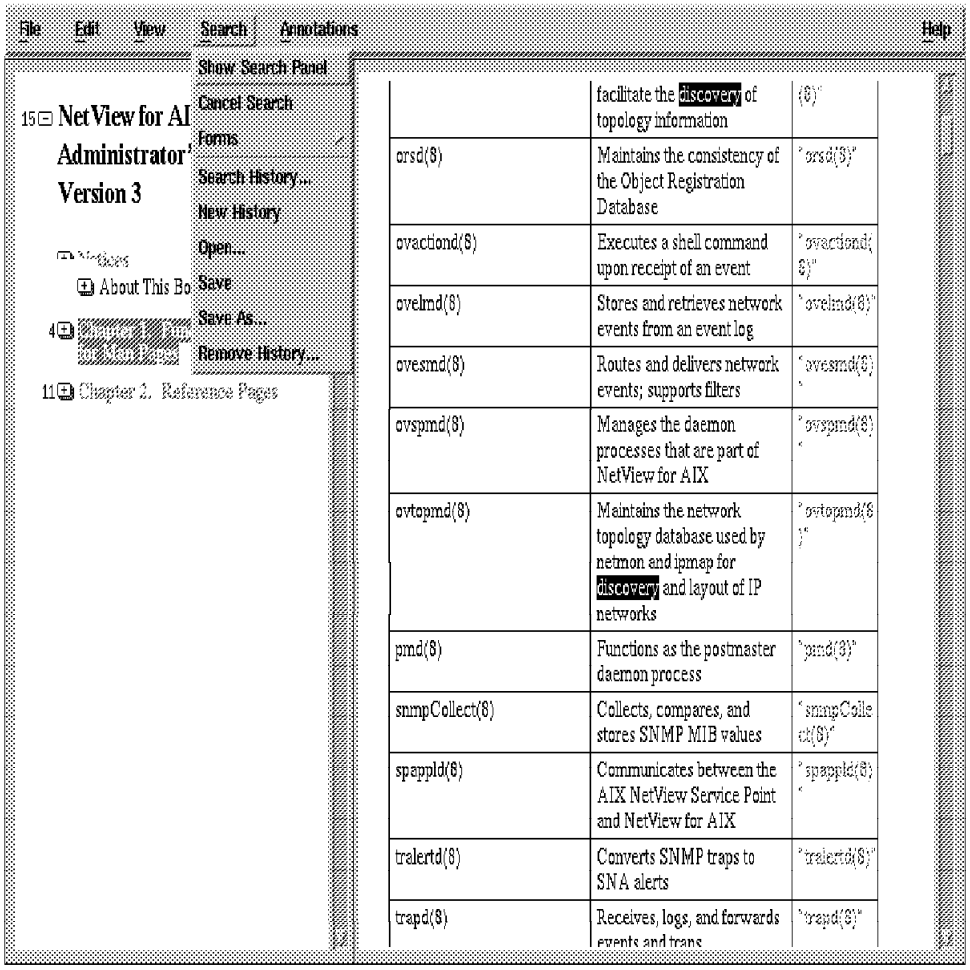


Figure 18. Setting Up Show Search Panel

Clicking on **Show Search Panel** results in being able to search various occurrences of the search operand via the buttons at the bottom of the following figure.

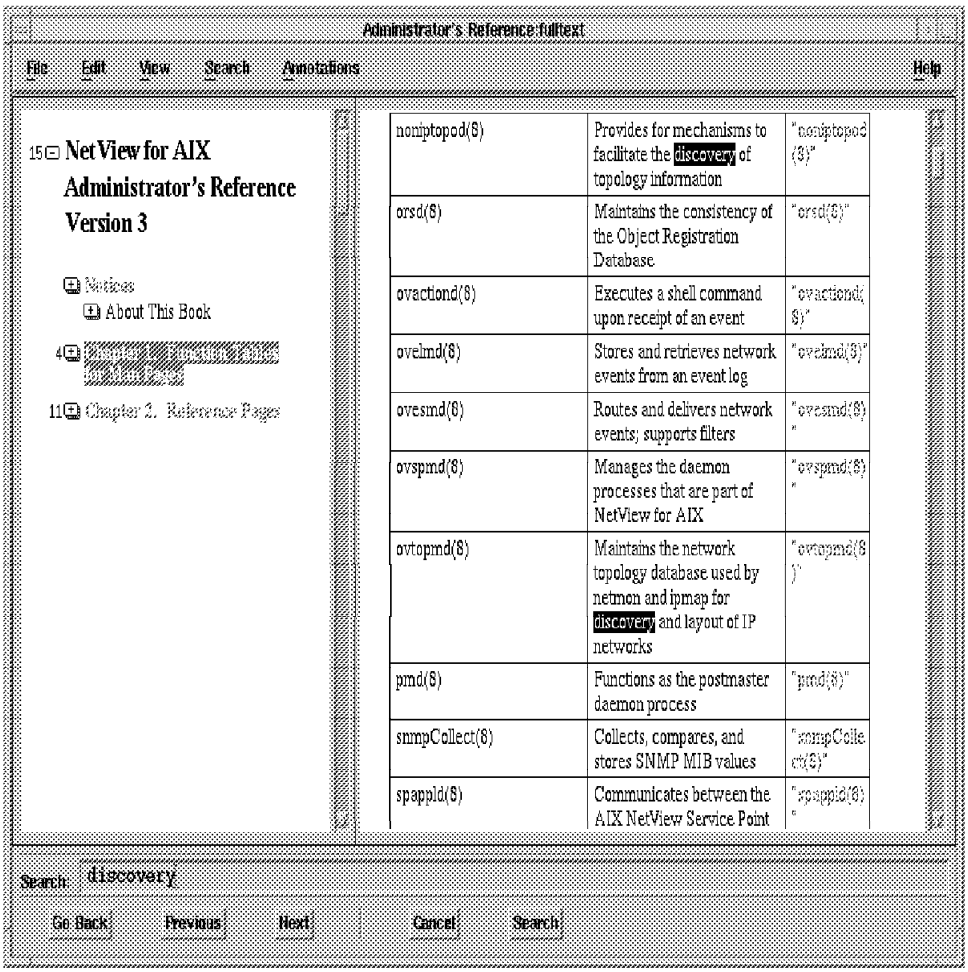


Figure 19. Search Panel for Fulltext of Selected Document

In this example, we use the search buttons to find the topic surrounding the discussion of discovery and turning netmon discovery on and off and end up with the information in the following figure.

Using online Help and keyword search can be of much value to an operator or application programmer involved with NetView for AIX.

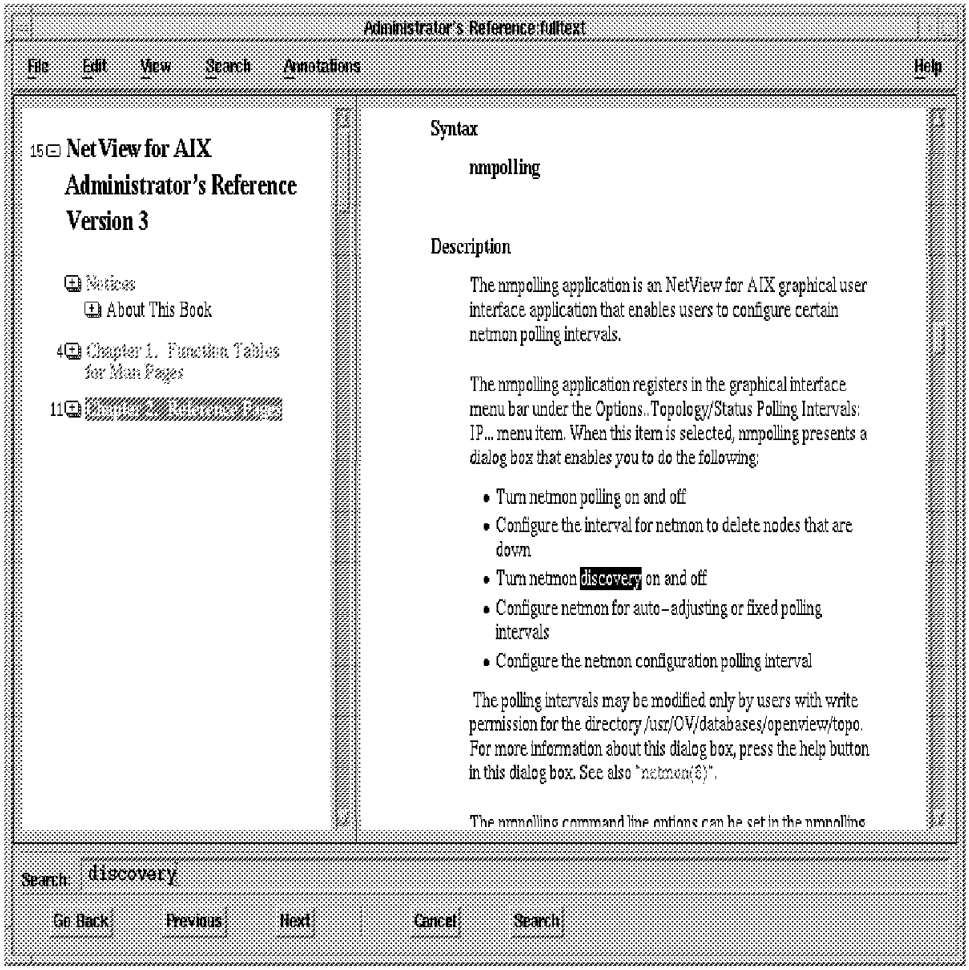


Figure 20. Search Panel for nmpolling Portion of Selected Document

2.8 Some Useful Hints

Following are some useful commands to use when the discovery process is not discovering certain devices on the network.

- `ovstatus netmon` checks the status of the discovery daemon.
- `getcommunity` `` `hostname` `` returns the local community name.
- `ping <hostname>` pings the device.
- `snmpwalk -c public nodename` steps through the MIB table for a particular node.
- `ovtopodump -v` lists the contents of the topology database.
- `ovobjprint -s nodename` reports the contents of fields for the nodename from the object database.
- `host nodename or IP address` resolves the hostname into an IP address or an IP address to hostname. Useful in name resolution diagnosis.

Chapter 3. Database Extensions

This chapter provides a summary of NetView for AIX database support. Refer to *IBM NetView for AIX Database Guide,* SC31-7190 for more information on this topic.

3.1 Overview of the Databases

Before discussing SQL relational databases in NetView for AIX V3R1, we should first explain how databases are architected in NetView for AIX.

NetView for AIX uses three different databases:

- A database to load SNMP data which has been collected via the snmpCollect command: the snmpCollect database.
- A database to collect alerts forwarded to S/390 NetView: the tralertd database.
- A database to store information about the managed instances: the openview database.

All these databases are located in the path /usr/OV/databases.

In addition, traps which are processed by NetView for AIX are located in a flat file called `trapd.log` in the directory /usr/OV/log. We mention this, since we will discuss moving trap data into SQL data later in this chapter.

3.1.1 SnmpCollect

The snmpCollect database is located in directory /usr/OV/databases/snmpCollect. Each data collection is placed in two files:

```
mibobjectname.i
mibobjectname.i!
```

Where `mibobjectname` is the name of the MIB variable being collected and `i` is the instance ID. The file suffixed ″!″ contains detailed information about the object and instance, plus some formatting information. The other file contains the collected data.

The data collected in these files is not readable by the user directly. It has to be extracted into a flat file using program `snmpColDump` or SMIT option NetView/6000->Control->Dump Data Collected by snmpCollect.

3.1.2 Tralertd

Database tralertd, located in `/usr/OV/databases/tralertd`, contains a copy of all the traps converted into an NMVT and intended to be sent to a S/390 NetView system. This database is internal to NetView for AIX and is not readable by the user.

3.1.3 The trapd.log File

The trapd database, file /usr/OV/log/trapd.log, contains all the traps received by trapd daemon. Information contained in this file is:

- Number of seconds since January 1, 1970 (midnight)
- Type of trap. This is a numeric field that represents the general category of the trap: Threshold, Topology, Error etc. You can find the possible values listed on the NetView for AIX Event Configuration panel.
- Date and time the trap was received, (in form: year-month-day time).
- Node which is the subject of the trap (note: not the origin of the trap, since some traps are generated internally by NetView for AIX to report problems found during network polling).
- Trap source. This is a single character that represents the type of agent that generated the trap. You can find the possible values listed on the NetView for AIX Event Configuration panel.
- Description. Textual description of the event indicated by the trap.

3.1.4 Openview

The ″openview″ database is, in fact, various sets of information. The end-user views this information primarily via the End User Interface (EUI) in graphic form. However, the data the user views has its base via a set of ″databases″.

The openview database is split into four separate files, which are described in Table 1.

Table 1. Usage of the Databases. The Openview files.

Database	Description
IP topology (IP)	The netmon process discovers an IP-capable device and provides IP topology information to the ovtopmd daemon. Ovtopmd stores the IP topology information in the IP topology database (which, in fact is many separate, related files) and creates IP objects in the ovwdb database.
gtm (non-IP)	A non-IP agent (such as LNM for AIX and AIX LMU/6000) sends non-IP topology information to the gtmd daemon. The gtmd daemon stores this information into the gtm database and, also, creates objects in the ovwdb database.
ovwdb	The ovwdb (sometimes known as the ″object″ database) contains the reference of all the instances which are created by netmon and the gtm daemon. In addition, user tasks create instances via NetView for AIX APIs.
map	Contains information resulting in graphical representation of the objects (symbols and submaps). It is filled by the ipmap application (IP Topology) using information from the IP topology database and by the xxmap application using information from the gtm database. In addition, other programs may create and maintain map objects via the NetView for AIX APIs.

Figure 21 on page 35 illustrates the relationships between these different components of the database, and the processes that bind them together.

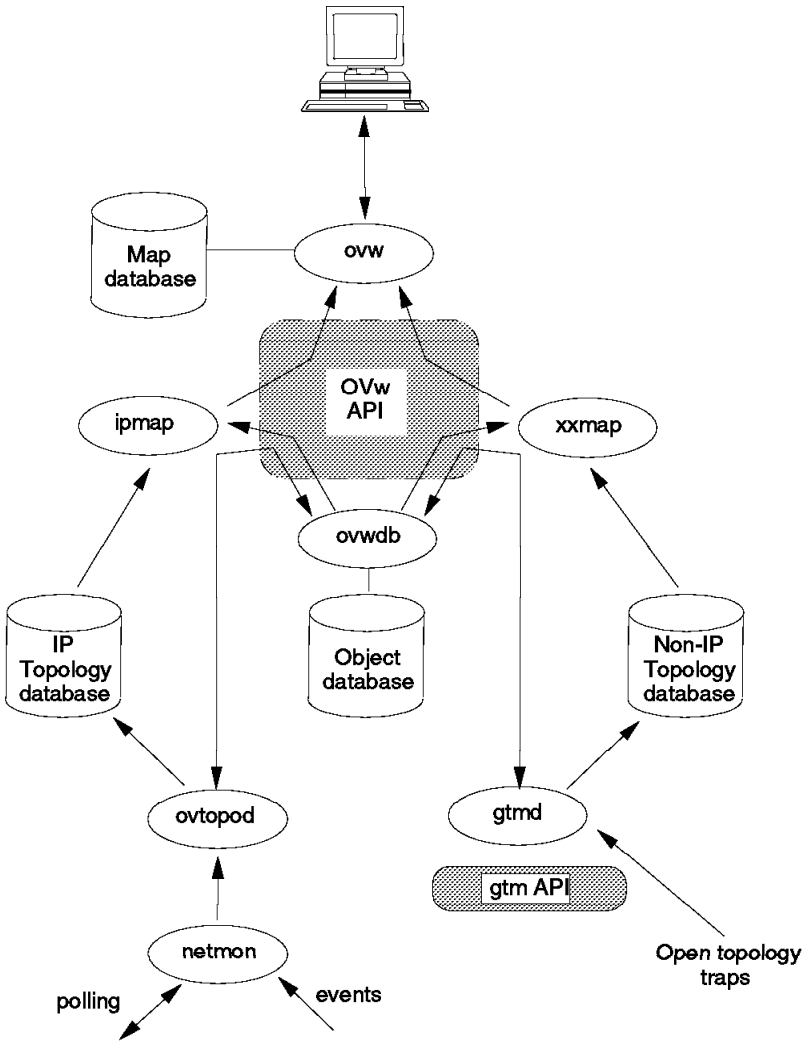


Figure 21. NetView for AIX Topology Databases and APIs

At base installation of NetView for AIX, the openview files are, by default, flat files contained in the directory /usr/OV/databases/openview.

Table 2 on page 36 lists the commands which extracts the main information from the openview databases. These commands do *not* require SQL database, they are normal commands distributed as part of NetView for AIX.

Table 2. List of the Commands	
Command	**Description**
ovtopodump	Returns the contents of the IP topology database. Information selected is: • Class, Object Id, Object, Status, IP Address
ovobjprint	Returns the contents of the ovwdb database. Information selected are fields related to an object: • Field Id, Field Name, Field Value
ovmapdump	Returns the contents of map database. Information selected is: • Submap • Submap id, Name, Policy, Parent, Layout • Object • Object id, Status, Compound status • Symbol • Symbol id, • Related to Object-id • Part of Submap-id • Variety (icon or connection)

Note

In AIX NetView/6000 V2R1, the IP topology database could be either in a flat file (default) or in an SQL database; however, only the Ingres SQL database could be used. In NetView for AIX this support has been extended to other SQL databases.

3.1.5 Extracting Information from the Flat File Database

Using commands `ovtopodump` (see Figure 22 on page 37), `ovobjprint` (see Figure 23 on page 38), and `ovmapdump` (see Figure 24 on page 39), we will illustrate the way to answer a question such as *"How is a particular node (we chose rs60001) configured from an IP point of view?"*. This is shown below. As a result we will be able to describe the node as shown in Table 3 on page 38. Later, we will make comparison with using SQL queries to answer the same question.

Figure 22 on page 37 illustrates the result of the `ovtopodump` command.

```
[rs60002:root] / > ovtopodump
CLASS     OBJECT ID      OBJECT                STATUS    IP ADDRESS
TOPOINFO     184         IP Internet
NETWORKS     189         9.24.104              Marginal  9.24.104.0
             227         9.67.32.64            Down      9.67.32.64
             229         9.67.38.64            Down      9.67.38.64
             231         9.67.46               Down      9.67.46.0
             237         9.24.1                Marginal  9.24.1.0
             239         9.24.96               Marginal  9.24.96.0
            5596         128.1.1               Unmanaged 128.1.1.0
            5052         150.0.49.32           Up        150.0.49.32
            5055         150.0.49.48           Up        150.0.49.48
            5065         150.0.49.16           Up        150.0.49.16
            5069         150.0.52              Up        150.0.52.0
            5077         150.0.49              Up        150.0.49.0
            5081         150.0.51              Up        150.0.51.0
SEGMENTS     190         9.24.104.Segment1     Marginal
             241         9.67.32.64.Segment1   Down
             242         9.67.38.64.Segment1   Down
             243         9.67.46.Segment1      Down
             244         9.24.1.Segment1       Marginal
             245         9.24.96.Segment1      Marginal
            5598         128.1.1.Segment1      Unmanaged
            5058         150.0.49.32.Segment1  Up
            5059         150.0.49.48.Segment1  Up
            5072         150.0.49.16.Segment1  Up
            5073         150.0.52.Segment1     Up
            5084         150.0.49.Segment1     Up
            5085         150.0.51.Segment1     Up
NODES        186/185     rs60002.itso.ral.ibm.com Up        9.24.104.28
             188/187     rs60003.itso.ral.ibm.com Up        9.24.104.23
             194 /193    rs60001.itso.ral.ibm.com  Up       9.24.104.26
             194 /228    rs60001.itso.ral.ibm.com  Down     9.67.32.85
             194 /230    rs60001.itso.ral.ibm.com  Down     9.67.38.71
             194 /232    rs60001.itso.ral.ibm.com  Down     9.67.46.29
             204/203     rs60005.itso.ral.ibm.com Up        9.24.104.25
             204/247     rs60005.itso.ral.ibm.com Down      9.67.32.86
             204/248     rs60005.itso.ral.ibm.com Down      9.67.38.67
             206/205     rs60004.itso.ral.ibm.com Up        9.24.104.27
```

Figure 22. Result of ovtopodump Command

From the result of this command, we can see, as an example, that node rs60001 which is rs60001.itso.ibm.ral.com (objid=194 in ovw database) has four interfaces.

To find out additional information regarding the interfaces, we can issue the command **ovobjprint** and request the information for a specific IP node as shown via the -s option:

```
[rs60002:root] / > ovobjprint -s rs60001.itso.ral.ibm.com
OBJECT:  194

 FIELD ID FIELD NAME          FIELD VALUE
 10       Selection Name      " rs60001 .itso.ral.ibm.com"
 11       IP Hostname         "rs60001.itso.ral.ibm.com"
 14       OVW Maps Exists     8
 15       OVW Maps Managed    8
 19       IP Status           Marginal(3)
 22       isIPRouter          TRUE
 29       vendor              IBM(1)
 39       isNode              TRUE
 41       isComputer          TRUE
 42       isConnector         TRUE
 43       isBridge            FALSE
 44       isRouter            TRUE
 45       isHub               FALSE
 48       isWorkstation       TRUE
 63       isIP                TRUE
 64       isSNMPSupported     TRUE
 66       SNMP sysDescr       "IBM RISC System/6000
                               Machine Type: 0x0010 Processor id: 000181471000
                               The Base Operating System AIX version: 03.02.0000.0000
                               TCPIP Applications version: 03.02.0000.0000 "
 67       SNMP sysLocation    ""
 68       SNMP sysContact     "Rob Macgregor's V3.2.5  Rob is in: BB112  x1-2325"
 69       SNMP sysObjectID    "1.3.6.1.4.1.2.3.1.2.1.1.2"
 70       SNMPAgent           IBM RS/6000(1)
 77       TopM Interface Count 4
 83       TopM Interface List "tr2; Up   9.24.104.26 255.255.255.0   0x 10005AA8D769  ieee 802.5 tokenRing"
                              "et0; Down 9.67.32.85  255.255.255.192 0x 02608C2EB97C  ieee 802.3 csmacd"
                              "tr0; Down 9.67.38.71  255.255.255.192 0x 10005AA88793  ieee 802.5 tokenRing"
                              "tr1; Down 9.67.46.29  255.255.255.192 0x 10005AC94085  ieee 802.5 tokenring"
 97       XXMAP Protocol List "IP"
 165      SMhasSysmon         TRUE
 191      IP Name             "rs60001.itso.ral.ibm.com"
 254      default IP Symbol List  14
                                  17
                                  18
                                  128
                                  129
                                  133
                                  134
                                  146
                                  147
```

Figure 23. Result of ovobjprint Command

With the information we have collected, and knowing how the IP subnet mask operates, we can build the following table:

Table 3. Node rs60001. Networks to which rs60001 is connected.

IP Interface	ovw Interface objid	IP Mask	IP Network	ovw Network objid	Physical Address
9.24.104.26	193	255.255.255.0	9.24.104	189	10005AA8D769
9.67.32.85	228	255.255.255.192	9.67.32.64	227	02608C2EB97C
9.67.38.71	230	255.255.255.192	9.67.38.64	229	10005AA88793
9.67.46.29	232	255.255.255.192	9.67.46	231	10005AC94085

To obtain the IP Network field above, we applied the network mask to the IP interface address.

The above information will be shown later (see Figure 34 on page 56) as the result of an SQL query.

The command ovmapdump will give us information about the relationship between symbols, submaps and objects. An example of information from ovmapdump is in Figure 24 on page 39 and provides the following information:

► 1 Object 194 (rs60001) is represented by symbol 14 in submap 2
► 2 Object 194 has an associated submap, the submap 13
► 3 Object 193 (rs60001:tr0;) is contained in submap 13, its parent (the node) is object 194 (► 2). That means interface rs60001:tr0; is part of node rs60001.

```
[rs60002:root] / > ovmapdump
     ELEMENT  MAP NAME                          PERMISIONS  CREATION TIME
     Map      default                           Read/Write  Fri Mar 18 08:36:38 1994

     ELEMENT  ID          NAME                 POLICY     PARENT     LAYOUT
     Submap   40          IPMap - Segment:9.67 Shared     243        Ring
     Submap   7           IPMap - Network:9.67 Shared     231        Point-to-Point
     Submap   2           IP Internet          Shared     184        Point-to-Point
     Submap   6           IPMap - Network:9.67 Shared     229        Point-to-Point
     Submap   10          IPMap - Segment:9.24 Shared     190        Ring
     Submap   41          IPMap - Segment:9.67 Shared     242        Ring
     Submap   5           IPMap - Network:9.67 Shared     227        Point-to-Point
     Submap   42          IPMap - Segment:9.67 Shared     241        Bus
► 2  Submap   13          rs60001              Shared     194        Row Column

     ELEMENT  ID          OBJECT STATUS        COMPOUND STATUS
     Object   194         Unknown              Marginal

     ELEMENT  ID          LABEL                OBJECT     SUBMAP     VARIETY
► 1  Symbol   14          rs60001              194        2          Icon
     Symbol   18          rs60001              194        10         Icon
     Symbol   128         rs60001              194        7          Icon
     Symbol   129         rs60001              194        40         Icon
     Symbol   133         rs60001              194        6          Icon
     Symbol   134         rs60001              194        41         Icon
     Symbol   146         rs60001              194        5          Icon
     Symbol   147         rs60001              194        42         Icon
► 3  Symbol   19          tr2;                 193        13         Icon
     Symbol   21          tr0;                 230        13         Icon
     Symbol   22          tr1;                 232        13         Icon
     Symbol   20          et0;                 228        13         Icon
```

Figure 24. Command ovmapdump

3.2 What Database Support is New in NetView for AIX?

NetView for AIX allows use of RDBMS (Relational DataBase Management Systems) for the following databases:

- IP Topology
- snmpCollect (MIB data collection)
- The event log (trapd daemon)

The relational database systems supported for these databases are:

- Ingres (IP Topology support for this was available in V2)
- DB2/6000
- Informix
- Oracle
- Sybase

The methodology of implementing RDBMS support in NetView for AIX differs with each particular NetView for AIX database (IP topology, trapd, and snmpcollect).

- IP topology SQL database

  Ovtopomd can store data either in a flat file (default option) *or* in a RDBMS. You make the choice when configuring the ovtopmd daemon. Performance considerations and usefulness of SQL reports are the two criteria for such a choice.

The first way to use RDBMS for ovtopmd is to make ovtopmd output its data directly into SQL tables (either local or remote database system). This is an online process.

The second way to use RDBMS for ovtopmd, is to make ovtopmd put its data into flat files (default option). Then, at a user-chosen time, convert data from flat files to SQL tables, this option could be automated using *"***cron***"* mechanism. This is an offline process.

A third way to use RDBMS for ovtopmd could be:

1. Use the flat file option and let ovtopmd discover the network.
2. Stop ovtopmd.
3. Convert the IP topology flat file into SQL tables.
4. Configure ovtopmd to use RDBMS.
5. Start ovtopmd.

Why would you do this? One reason would be to improve performance during the discovery process.

- trapd and snmpCollect

  Trapd and snmpCollect continue to use flat files for logging, as for previous NetView for AIX releases. The RDBMS support in NetView for AIX allows conversion of trapd and snmpCollect operational data from flat files to SQL tables for further analysis using SQL facilities. This is an offline process. For further information, refer to *IBM NetView for AIX Database Guide,* SC31-7190.

3.3 Configuration Steps

We tested the SQL database environment with both Oracle and Informix RDBMS.

For Oracle IP topology SQL support, we tested using both a local and a remote data server configuration. For Informix IP topology SQL support, we tested only a local data server configuration.

Note

AIX SystemView NetView/6000 Database Guide, SC31-7190 recommends using NetView for AIX with a remote database server rather than with a local database server for performance reasons.

Note that although client/server database configurations are supported this does *not* mean that multiple NetView for AIX images may share one database.

We will look at some examples of supported and unsupported configurations. Figure 25 on page 41 shows a supported remote server RDBMS.

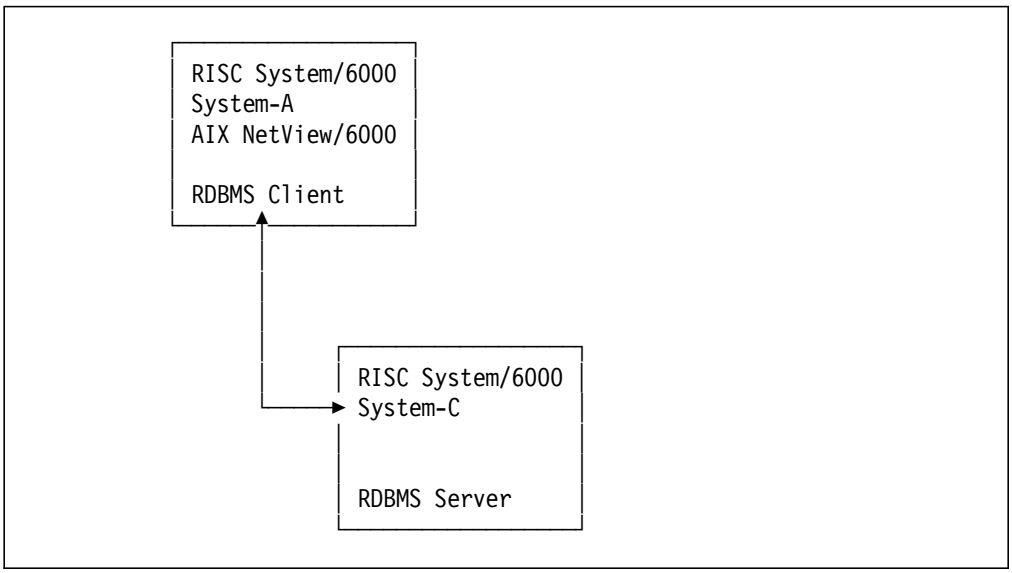


Figure 25. Supported Remote Server RDBMS

The IP Topology SQL tables cannot be shared by multiple clients, nor can a single database server have multiple copies of the databases. So the configuration in Figure 26 on page 41 would not be valid.

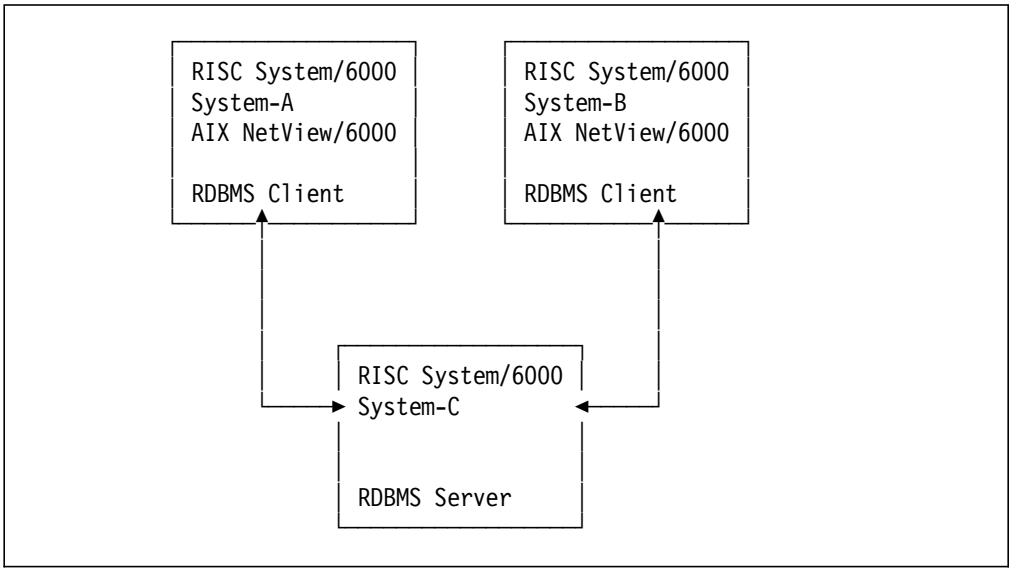


Figure 26. Unsupported Remote Server RDBMS

Note, however, that the snmpCollect and event log database tables can contain merged data from different NetView for AIX sources. For IP Topology, each client would need its own server:

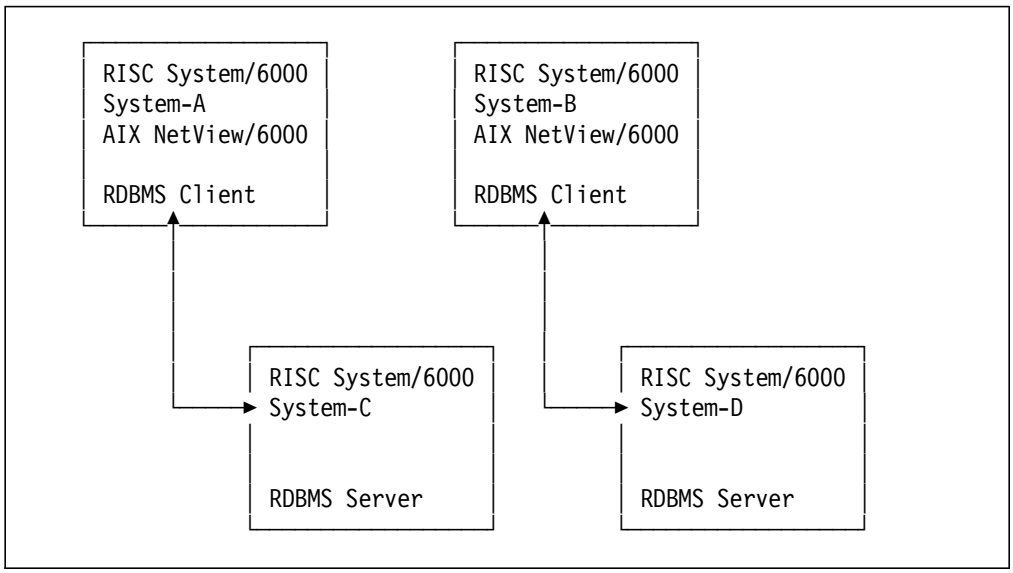


Figure 27. Supported Remote Server RDBMS

Using RDBMS within NetView for AIX requires some knowledge and/or access to database administration. This important matter cannot be ignored. For example, configuration changes must be made in agreement with the database administrator:

- Size of the database
- Recovery matters and other considerations such as how to resolve exceptional situations such as filling up the SQL database

At least five steps are necessary before starting to use the RDBMS in NetView for AIX:

1. Configure the AIX system for use of the RDBMS.
2. Create the database.
3. Specify which RDBMS is the default one for NetView for AIX.
4. While configuring NetView for AIX, specify whether ovtopmd (IP topology database daemon) will use SQL tables or flat files (default).

   trapd and snmpCollect are not configured as SQL, since the RDBMS support is an offline process.
5. Create the SQL tables.

You should refer to the particular RDBMS installation and customization documentation and have access to *AIX SystemView NetView/6000 Database Guide,* SC31-7190 prior to the following steps.

3.3.1 Configuration of the AIX System for Use of RDBMS

Assuming that the RDBMS is already installed on the system, you have to check the following three points to use RDBMS with NetView for AIX:

1. Set the environment variables on the client and server systems, if both are being used. Otherwise, set the environment variables only the server.

   Figure 28 on page 43 shows the additions we made to `$HOME/.profile` to set the environment variables needed for the two databases we used (using Korn shell).

```
# oracle database
# SID=I because of Transmission Network Manager/6000 database
export ORACLE_SID=I
export ORACLE_HOME=/usr/oracle
export PATH=$PATH:$ORACLE_HOME/bin
# informix database
export INFORMIXDIR=/usr/informix
export SQLEXEC=$INFORMIXDIR/lib/sqlturbo
export PATH=$PATH:/$INFORMIXDIR/bin
```

Figure 28. Setting the AIX Environment Variables. These variables would be set on both client and server.

Note

The sqlturbo refers to the Informix Online Server only.

2. Add port definitions to /etc/services to allow remote database access (only if the server is on a separate machine). Refer to the documentation for the particular database being used.
3. Start the database daemons in the AIX system (refer to your database documentation for details).

3.3.2 Create the Database

Using the create command, specific for each RDBMS, create the database in agreement with your database administrator's specifications.

In our example we had already installed Oracle for AIX Transmission Network Manager/6000. Thus we used the same physical database for both products; we did not have to create a specific database for NetView for AIX.

For Informix, we created a database using the dbaccess command. We had to create it with log mode ANSI as shown:

```
dbaccess - -
> create database openview with log mode ansi;

Database created

>
```

3.3.3 Specifying Default RDBMS System

Our second step is to define which default RDBMS will be used by NetView for AIX. This is done by filling in the screen found under NetView for AIX SMIT menus in: Configure → Configure relational database.

The database entered here will be the default for all requests for which a Relational Database Management System is used. For example, when we later issue the command to archive snmpCollect data using the nvColToSQL command, we do not need to specify the RDBMS in use.

In this example we configure NetView for AIX (running on machine rs60002) to use remote database server on machine rs60001. In order to achieve this

configuration, *orasrv* (Oracle communication) has to be started on machine rs60002.

Note that "SID=I" allows NetView for AIX to share the same database as the one used by TNM/6000:

```
                          Configure relational database

Type or select values in entry fields.
Press Enter AFTER making all desired changes.
                                          [Entry Fields]
Database name:                            oracle                        +
Database server name:                     [rs60001]
SID (only for Oracle database):           [I]
log file:                                 [/usr/OV/log/dblog]
Trace file:                               [/usr/OV/log/dbtrace]
Tracemask:                                [1]                           #
```

The second example (below) shows how to configure NetView for AIX to use Informix in a local configuration. Note that the Database server name field is left blank signifying a local server.

```
                          Configure relational database

Type or select values in entry fields.
Press Enter AFTER making all desired changes.
                                          [Entry Fields]
Database name:                            informix                      +
Database server name:                     []
SID (only for Oracle database):           []
log file:                                 [/usr/OV/log/dblog]
Trace file:                               [/usr/OV/log/dbtrace]
Tracemask:                                [1]                           #
```

3.3.4 Specify that ovtopmd Will Use a Relational Database

This step can be skipped if the flat files (default) are to be used by the daemon ovtopmd.

Note

This option was already existing in Version 2. In Version 2, the RDBMS could only be Ingres. In Version 3, the Relational Database will be the one selected in the previous step.

This will be found under NetView for AIX menus: Configure → Set options for topology, discovery, and database daemons →Set options for ovtopmd.

```
                      Set Options for ovtopmd daemon

Type or select values in entry fields.
Press Enter AFTER making all desired changes.
                                          [Entry Fields]
Do you want to use an SQL relational database? [yes]
```

3.3.5 Creation of SQL Tables in Openview Database

The last step is the creation of the SQL tables for the three different uses:

1. IP Topology data
2. traplog data
3. snmpCollect data

Note that all the tables reside in the one database, named ″openview″.

3.3.5.1 Creating IP Topology SQL Tables

Table 4 shows specific command to create SQL tables depending of which RDBMS is used. To create the SQL table issue the specific command with root authority. Using the default parameters set up during step 1: 3.3.3, “Specifying Default RDBMS System” on page 43, only the flags -c and -v are necessary.

Table 4. Commands for Creation of SQL Tables

Database	Command	Flags
DB2/6000	db2_ovtoposql	-c -v
Informix	in_ovtoposql	-c -v
Ingres	ovtoposql	-c -v
Oracle	oc_ovtoposql	-c -v
Sybase	bin/sy_ovtoposql	-c -v

Using Oracle, the command to issue is oc_ovtoposql -c -v.

```
[root:rs60002] / > oc_ovtoposql ?
Usage: oc_ovtoposql {-c|C|D} [-v] [-U DB userid [-P DB password]] [-S server]
        -c              Create the topology Oracle SQL tables.
        -C              Clear the topology Oracle SQL tables.
        -D              Drop the topology Oracle SQL tables.
        -v              Verbose mode.
        -U DB userid    Oracle database owner login id.
        -P DB password  Oracle database owner login password.
        -S server       Server connect string <server_node:ORCALE_SID>
```

Note that the same command will be used to clear the database (-C option) or to drop it (-D option).

```
[root:rs60002] > in_ovtoposql -c -v
Using default database openview.
Connecting to database openview.
Creating table networkclass.
Creating unique index xnetworkclass on networkclass.
Creating table segmentclass.
Creating unique index xsegmentclass on segmentclass.
Creating table nodeclass.
Creating unique index xnodeclass on nodeclass.
Creating table interfaceclass.
Creating unique index xinterfaceclass on interfaceclass.
Creating table topoinfo.
Creating table classtable.
Creating table objectable.
Creating unique index xobjectable on objectable.
Creating index yobjectable on objecttable.
Creating table memberof.
Creating index xmemberof on memberof.
Creating index ymemberof on memberof.
Creating table coupledwith.
Creating index xcoupledwith on coupledwith.
Creating index ycoupledwith on coupledwith.
Initializing topology class table.
Granting access to database tables to all users.
Closing database.
Done.
```

3.3.5.2 Creating trapdlog SQL Table

To create the trapd SQL table issue the command trapsql with root authority. Using the default parameters set up during step 1 3.3.3, “Specifying Default RDBMS System” on page 43, only the flags -c -v are necessary.

```
[root:rs60002] / > trapsql ?
Usage: trapsql [-c|-C|-D][-v][-h][-l logfile][-t   tracefile]
               [-m tracemsk][-Z dbtype][-U DBA userid][-P DBA password][-S server]
        -c              Create the trapd SQL tables.
        -C              Clear the trapd SQL tables.
        -D              Drop the topology SQL tables.
        -v              Verbose mode.
        -h              Help.
        -l logfile      Change log file name.
        -m tracemask    Trace mask values. Options are: 0 Tracing off,
                        1  Insert trace, 2  Select trace, 4 Update trace,
                        8  Delete trace, 16  Misc trace, 32 Entry trace,
                        64 Error trace, 127 All traces
        -t tracefile    Change trace file name.
        -Z dbtype       Type of SQL database. Options are: Ingres,
                        Oracle, Sybase, Informix and DB2.
        -U DBA userid   Database Administrator login id.
        -P DBA password Database Administrator login password.
        -S server       SQL Server name.
```

Note that the same command will be used to clear the database (-C option) or to drop it (-D option).

```
[root:rs60002] / > trapsql -c -v
Connecting to Database Server.
Creating table traplogd.
Trapsql utility complete.
```

3.3.5.3 Creating snmpCollect Tables

To create the snmpCollect SQL table issue the command nvColTable with root authority. Using the default parameters set up during step 1 3.3.3, “Specifying Default RDBMS System” on page 43, only the flags -c -v are necessary.

```
[root:rs60002] / > nvColTable ?
Usage: nvColTable {-c|-C|-D} [-Z dbtype] [-l logfile] [-t tracefile] [-m tracemask]
                  [-U DBA Id [-P DBA password]] [-S servername] [-v] [-h]
        -c              Create the snmpCol SQL tables.
        -C              Clear the snmpCol SQL tables.
        -D              Drop the snmpCol tables.
        -Z dbtype       Type of SQL database. Options are:
                        Ingres, Oracle, Sybase, Informix and DB2
        -l logfile      User specified log file
        -t tracefile    User specified trace file
        -m tracemask    Trace mask values. Options are:
                        1  Insert trace, 2  Select trace, 4 Update trace
                        8  Delete trace, 16  Misc trace, 32 Entry trace
                        64 Error trace, 127 All traces
        -U DBAId        Database Administrator login Id.
        -P DBApassword  Database Administrator login password.
        -S servername   SQL Server name
        -v              Verbose mode
        -h              Help
```

Note that the same command will be used to clear the database (-C option) or to drop it (-D option).

```
[root:rs60002] / > nvColTable -c -v
Connecting to Database Server.
Creating table colData.
Creating table varInfo.
Creating table expInfo.
nvColTable utility complete.
```

3.4 Using IP topology SQL Tables

Figure 29 on page 48 shows the four databases that comprise the openview database (note that **FF** stands for ″Flat File″).

1. ovw database (Flat File only):
   - Managed by ovwdb daemon
   - Queried by ovobjprint standard command
   - Note that the wtovwconv program is a sample developed as part of this project to give the ability to load these data in an SQL table (discussed in 3.4.1, “Structure of IP Topology SQL Tables” on page 49).
2. IP topology database (Flat File or SQL database):
   - Managed by ovtopmd daemon

- Queried by ovtopodump command if stored in a flat file file
- Any SQL query if using an SQL database (such as the commands and routines described later in this chapter).

3. Non-IP topology database (Flat File only):
   - Managed by gtmd daemon
   - No standard command to access the data (note that the Open technology API will allow such a command to be produced)
4. Map database(s) (Flat File only):
   - Managed by ipmap and xxmap applications
   - Queried by ovmapdump command (and the NetView for AIX GUI)

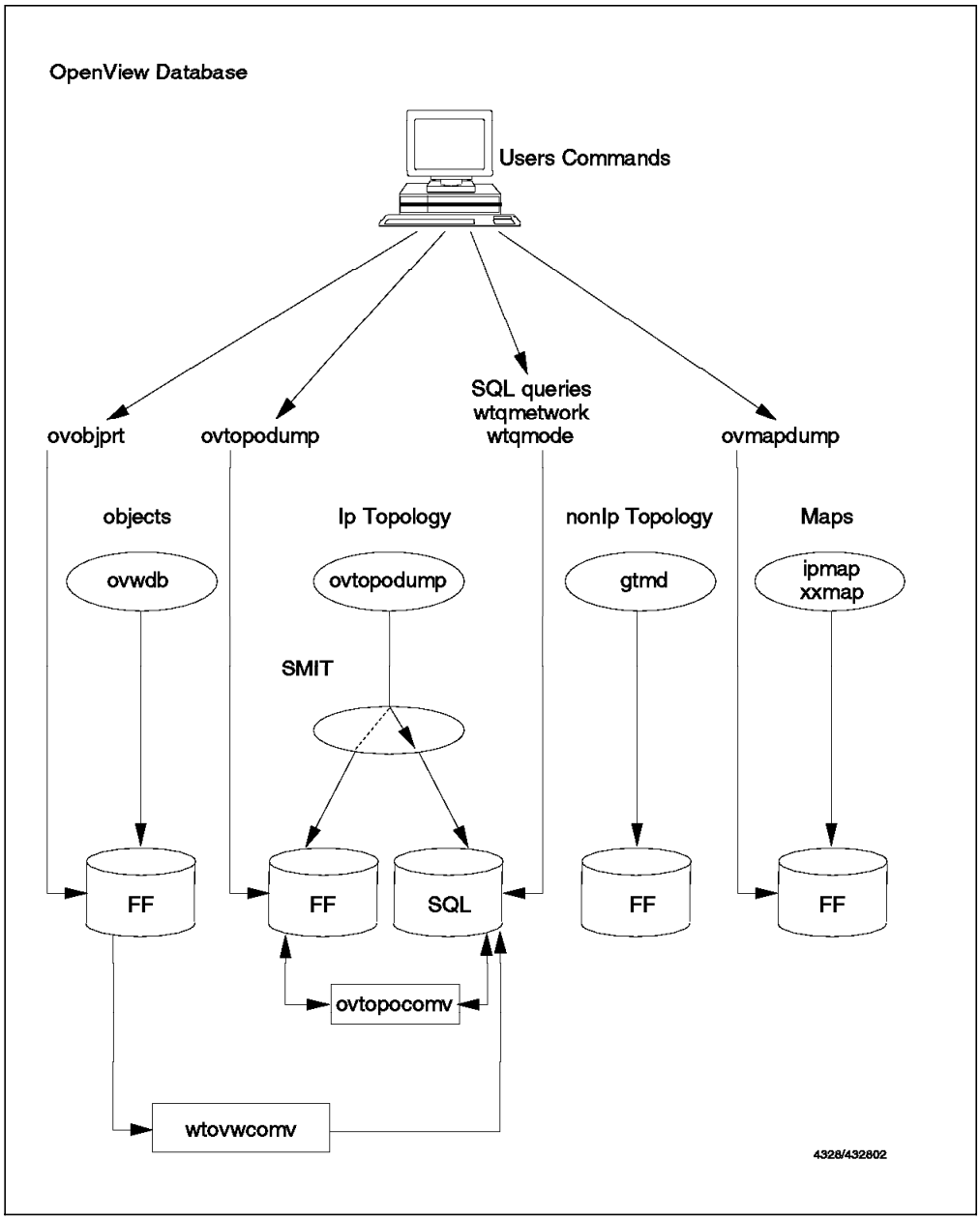


Figure 29. The OpenView Database

3.4.1 Structure of IP Topology SQL Tables

Table 5 gives a complete list of the SQL tables for the IP topology database.

Table 5 (Page 1 of 2). IP Topology Database. Description of the SQL tables.

Table	Description
networkclass	Contains information about network, each row represents one network in the IP topology. • objid • ip_network_name, ip_status • ip_address, ip_subnet_mask • topm_interface_count, topm_segment_count
segmentclass	Contains information about segment, each row represents one segment in the IP topology. • objid • selection_name, ip_status • topm_interface_count
nodeclass	Contains node information, each row represents one node in the IP topology. • objid • ip_hostname, ip_status • snmp_sysdescr, snmp_syslocation, snmp_syscontact • topm_interface_count • snmp_sysobjid, ipforwarding • snmpaddr
interfaceclass	Contains interface information, each row represents one interface in the IP topology. • objid • snmp_ifdescr, ip_status • ip_address, ip_subnet_mask • ifnumber, snmp_iftype • snmp_ifphysaddr
classtable	This table describes each object class, it is composed of two columns ***classid*** and ***name***, and it contains four rows: • classid name • 1 network • 2 segment • 3 node • 4 interface
objecttable	This table describes to which classid (described in the classtable) belongs each object contained in networkclass, segmentclass, nodeclass or interfaceclass tables. The objecttable is composed of the two columns: • objid • classid (1,2,3,4)
memberof	This table contains the two relationships, **interface member-of node** and **segment member-of network**. The table is composed of the two columns: • containedobjid • containerobjid
coupledwith	This table contains the two relationships, **interface coupled-with segment** and **interface coupled-with network**. The table is composed of the two columns: • objid1 • objid2 **Note:** To improve performance, each relationship appears twice in the table with the IDs swapped between the two columns.

Table 5 (Page 2 of 2). IP Topology Database. Description of the SQL tables.	
Table	**Description**
topoinfo	Each column of this table is one specific counter (number of objects, number of nodes, etc...). This table contains just one row; the up-to date counters.

3.4.2 Conversion Between Flat File and SQL Database

Depending on the option used, the same command ovtopoconv can be used to convert flat files to SQL tables or to convert SQL tables to flat files for the IP topology database.

```
[root:rs60002] ovtopoconv ?
Usage: ovtopoconv {-S|I|R|N|F|A|B|C|D|E} [-v]
        -S      Convert to the flat file from Sybase RDBMS.
        -I      Convert to the flat file from Informix RDBMS.
        -R      Convert to the flat file from Oracle RDBMS.
        -N      Convert to the flat file from Ingres RDBMS.
        -F      Convert to the flat file from DB2/6000 RDBMS.
        -A      Convert to the Sybase RDBMS from flat file .
        -B      Convert to the Informix RDBMS from flat file .
        -C      Convert to the Oracle RDBMS from flat file .
        -D      Convert to the Ingres RDBMS from flat file .
        -E      Convert to the DB2/6000 RDBMS from flat file .
        -v      Verbose mode.
```

3.4.2.1 Convert IP Topology Data from Flat File to SQL Database

This option is used when ovtopmd has been configured to run with flat files. It allows you to take a snapshot of the IP Topology database in order to make an SQL query. This option could be automated using the *cron* mechanism. The conversion needs to have empty database tables to write into, so it has to be preceded by SQL table cleanup. Figure 30 shows a sample script file that performs the sequence.

```
#check if tbinit (informix server is running)
TBINIT=`ps -ef | grep -e grep | grep tbinit | wc -l`
if [ "$TBINIT" -ne 1 ]
   then tbinit
fi

# using Informix
# 1. Clean the database
in_ovtoposql -C -v

# 2. Make the conversion
ovtopoconv -B -v
```

Figure 30. Script file to Automate Conversion of IP Topology Data in SQL Tables

The result of running this command is shown below:

```
Using default database openview.
Connecting to database openview.
Truncating  the table networkclass.
Truncating the table segmentclass.
Truncating the table nodeclass.
Truncating the table interfaceclass.
Truncating the table topoinfo.
Truncating the table classtable.
Truncating the table objecttable.
Truncating the table memberof.
Truncating the table coupledwith.
Re-adding  records into class table.
Closing database.
Done.
Using default database name openview.
Converting from private datastore to SQL.
Verifying that there is no ovtopmd executing.
Opening source database
Opening target database
Converting global topology information.
Converting 9 networks.
    1/9 9.67.46.128.
    ...
    9/9 9.24.104.
Converting 9 segments.
    1/9 9.67.46.128.Segment1.
    ...
    9/9 9.24.104.Segment1.
Counting Nodes. Please wait : This will take some time ...
Converting 30 nodes.
    1/30 eamon.itso.ral.ibm.com.
   .....
   30/30 rs60004.itso.ral.ibm.com.
Closing databases.
Done.
```

Note

The daemon ovtopmd has to be stopped, using command ovstop ovtopmd prior to running the conversion.

3.4.2.2 Convert IP Topology Data from SQL Database to Flat File

This option could be used when it has been decided to get back to the configuration of flat file for ovtopmd daemon.

3.5 Examples of SQL Queries for the IP Topology Database

In this section we explain how to use SQL to make queries of the IP topology database. As the IP topology database is comprised of nine tables, we will want information binding more than one class of object (network, segment, interface or node). To achieve this we must *join* tables and and use SQL subquery.

Our objective is to use SQL to extract the information that we previously extracted using ovobjprint and ovtopoprint (see 3.1.5, “Extracting Information from the Flat File Database” on page 36) and then to present that information in a simple way.

The queries we construct will provide, for one or several nodes, the characteristics of the nodes, including:

- Characteristics of the node itself.
- List of interfaces with their own characteristics.
- For each interface the network to which it belongs.

Figure 31 shows the first query needed, to give in one shot, information about the node itself and the associated interfaces.

To achieve that, we have to join tables:

1. nodeclass returns node information,
2. memberof returns list of the interface IDs belonging to this node.

The joining of these tables is performed by,

- select from nodeclass,memberof allows us to query concurrently the two tables nodeclass and memberof.
- nodeclass.objid=memberof.containerobjid allows us to select only rows where the container (in memberof table) and the node (in nodeclass table) have the same object-id.

Note that the condition ip_hostname like '%string%' allows query of one node or several nodes (when ip_hostname matches with "string"), or all the nodes when string is blank.

```
select nodeclass.objid,ip_hostname,snmpaddr,containedobjid
  from nodeclass,memberof
  where
    containerobjid =
    (select objid from nodeclass where ip_hostname like '% rs60001 %')
  and
    nodeclass.objid=memberof.containerobjid
  ;
```

Figure 31. SQL Query of Interfaces for NodeID=rs60001

The above query lists all the interface objID for node rs60001. The resulting display looks like this:

```
objid           194
ip_hostname     rs60001.itso.ral.ibm.com
snmpaddr        9.24.104.26
containedobjid  193

objid           194
ip_hostname     rs60001.itso.ral.ibm.com
snmpaddr        9.24.104.26
containedobjid  228

objid           194
ip_hostname     rs60001.itso.ral.ibm.com
snmpaddr        9.24.104.26
containedobjid  230

objid           194
ip_hostname     rs60001.itso.ral.ibm.com
snmpaddr        9.24.104.26
containedobjid  232

objid           194
ip_hostname     rs60001.itso.ral.ibm.com
snmpaddr        9.24.104.26
containedobjid  6015
```

The query above gave us the list of the objIDs of all interfaces in a specific node. We have now to issue, for each interface-id returned, a query such as in Figure 32 on page 54, giving in one shot information about the interface and its network. In this query we again join tables:

1. `interfaceclass`: returns interface information such as:
   - IP Address
   - Physical address
   - SNMP description
   - protocol (9 means IEEE 802.5)
2. `coupledwith`: returns network objID and segment objID which define the places within the topology where the interface is connected.
3. `objecttable`: allows us to restrict the result of table `coupledwith` to network only (`objecttable.classid`).
4. `networkclass`: returns network information, such as the IP Network name.

In order to get network information we have to issue a *sub-query* which joins:

1. `coupledwith`: returns the objectIDs of the network and segment that the interface belongs to.
2. `objecttable`: allows us to restrict the result of table `coupledwith` to network only.

```
select interfaceclass.objid,
       interfaceclass.ip_address,
       snmp_ifphysaddr,snmp_iftype,snmp_ifdescr,
       ip_network_name
       from interfaceclass,coupledwith,objecttable,networkclass
       where interfaceclass.objid= 193
       and coupledwith.objid2=interfaceclass.objid
       and coupledwith.objid1=objecttable.objid
       and objecttable.classid=1
       and networkclass.objid =
        (select objid1
          from coupledwith,objecttable
          and coupledwith.objid2= 193
          and classid=1
          and coupledwith.objid1=objecttable.objid);
```

Figure 32. Characteristics of Interface objid=193

The above example is a query that extracts the details of the interface whose objID is 193. The result is:

```
objid            193
ip_address       9.24.104.26
snmp_ifphysaddr  0x10005AA8D769
snmp_iftype      9
snmp_ifdescr     tr2; Product: not available! Manufacturer: not
available! Part Number: not available! FRU Number: not available!
ip_network_name  9.24.104
```

3.6 Combining and Formatting SQL Queries

The sample queries shown above extract the data we are interested in, but we would like to be able to link them together, and then format the results.

To do this we need to write a program that issues the initial query and then processes the subsequent queries for each interface that it returns.

There are two programming methods available to us:

- AIX script program:

  This program is developed as an AIX script command file. The main script program executes each SQL query in turn by connecting to the RDBMS,and executing the initial SQL query. The result is spooled to a file or pipe. Then the AIX script command explodes the output of the query and executes the subsequent queries.

- C language program:

  Using a C language program we can use more elaborate techniques, such as the *SQL cursor* mechanism. The SQL cursor allows processing of SQL statements that return more than one row. The programmer has to declare a cursor for the query, then he can proceed row-by-row to execute the subsequent queries.

We will illustrate this with two examples:

wtqnode Query a node or a list of nodes, developed using both of the mechanisms just described.

wtqnetwork Query a network or a list of networks, developed only with the SQL cursor (C program) mechanism.

3.6.1 SQL Sample wtqnode

The logic followed by both the shell script form and the C form of this program is identical:

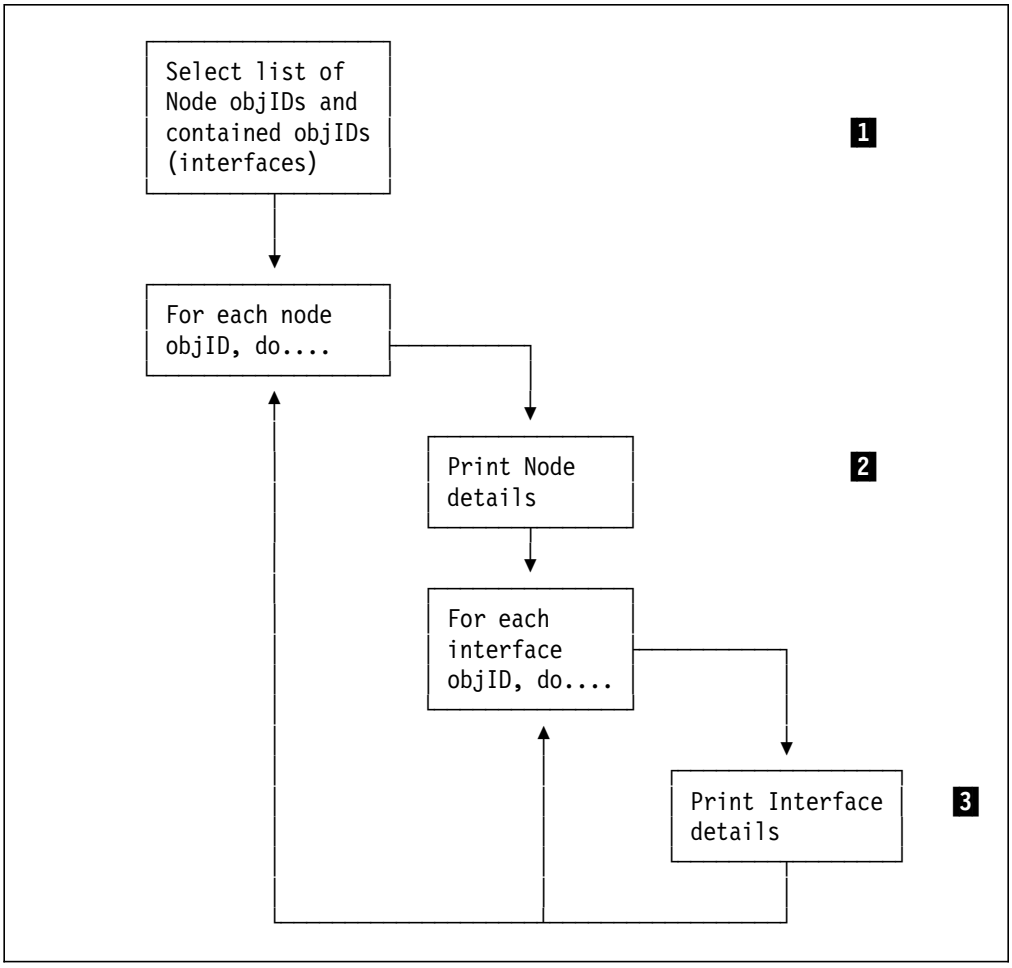


Figure 33. Logic for wtqnode Samples

Clearly, actions 1 and 2 are performed by the first SQL query we produced (see Figure 31 on page 52) and 3 is performed by the second (Figure 32 on page 54).

3.6.1.1 wtqnode - Shell Script Version

The source code for this and the following examples is listed in Appendix D, “Database Samples” on page 271.

Figure 187 on page 272 shows the AIX script file `wtqnode` which uses the Oracle RDBMS. You will note the `sqlplus -v uid/pwd @cmd` command within the script. This makes a connection on Oracle and executes the SQL commands contained in file `cmd.sql`.

As you would expect from the logic diagram above, there are three such SQL queries imbedded in the wtqnode script:

lc.sql For a given node name, writes in a file (standard output, redirected to wtqnode.out) the objID for each interface (see Figure 188 on page 272).

lo.sql For a given node objID, which has been read from the previous output file, print the value of an attribute passed as parameter (here ip_hostname and snmpaddress), (see Figure 189 on page 273).

li1.sql Gives the characteristics of a given interface object ID (see Figure 190 on page 273).

Note that in the .sql command files we are using:

- Positional arguments (&1, &2) to pass variable information to the embedded query
- The column command which allows us to resize the output

The rest of the script consists of wizardry from awk and cut to reformat the raw SQL query into something more readable.

An example of the result of the wtqnode shell script is shown below:

```
/ > wtqnode rs60001

IP hostname:
rs60001.itso.ral.ibm.com
IP address:
9.24.104.26
============================================================
ObjID    IP Address      Phys.Address    Type IP Network

193      9.24.104.26     0x10005AA8D769  tr2; 9.24.104
228      9.67.32.85      0x02608C2EB97C  et0; 9.67.32.64
230      9.67.38.71      0x10005AA88793  tr0; 9.67.38.64
232      9.67.46.29      0x10005AC94085  tr1; 9.67.46
```

Figure 34. Sample Result of wtqnode Shell Script. The node we are querying (rs60001) has four IP network interfaces.

3.6.1.2 wtqnode - C Program Version

Figure 191 on page 274 shows the source code for the C language version of wtqnode, written for use with the Informix RDBMS. In fact, this program source is a file of type .ec which indicates an embedded SQL program. This means that it has to be preprocessed using the esql command to expand the SQL queries before being processed by the regular "C" compiler. The make file used to do this is shown in Figure 192 on page 277.

The wtqnode command makes use of the SQL *cursor* mechanism which allows a program to manipulate data coming from a multi-line SQL query. This allows us to structure the queries a little better than in the shell script version. The first query (defined as q1) extracts the node information. The $fetch command then allows us to process each row returned in that query, by referencing the cursor, c1, which we previously declared for it.

The result of running this version of wtqnode against node rs60001 is shown below:

```
/ > wtqnode rs60001
======================================================================
objid:194 - ip_hostname: rs60001.itso.ral.ibm.com
======================================================================
Description : IBM RISC System/6000
Machine Type: 0x0010 Processor id: 000181471000
The Base Operating System AIX version: 03.02.0000.0000
TCPIP Applications version: 03.02.0000.0000
Location    :
Contact     : Rob Macgregor's V3.2.5  Rob is in: BB112  x1-2325
SNMP address: 9.24.104.26
Objid    IP Address      Network Name    Physic. Address Type Status
-------- --------------- --------------- --------------- ---- ------
193      9.24.104.26     9.24.104        0x10005AA8D76   tr2; up
228      9.67.32.85      9.67.32.64      0x02608C2EB97   et0; down
230      9.67.38.71      9.67.38.64      0x10005AA8879   tr0; down
232      9.6 7.46.29     9.67.46         0x10005AC9408   tr1; down
======================================================================

1 row(s) retrieved.
```

Figure 35. Example of wtqnode C Program Output. The C version of the routine displays more detailed information than the shell script. It also executes noticeably faster.

3.6.2 SQL Sample wtqnetwork

This sample extracts summary information about every node in an IP network or subnetwork. It allows you to specify a network address, or a pattern or to list all networks.

To do this, four nested SQL queries have to be embedded into the program:

1. List all the networks matching the pattern.
2. Within each network, list all the segments.
3. Within each segment, list the object ID of every interface.
4. List the details of a given interface object ID.

The source code (using Informix SQL embedded in 'C') is in Figure 193 on page 278. The result of running the program with a pattern of "9.24.1" is shown in Figure 36 on page 58.

```
/ > wtqnetwork 9.24.1
================================================================================
OVw id = 517 - Network name = 9.24.104
IP Address = 9.24.104.0      - Subnet mask = 255.255.255.0
--> Contents of Segment: 9.24.104.Segment1
--------------------------------------------------------------------------------
         ObjID  Stat IP Address           Phys. Address  Node
--------------------------------------------------------------------------------
     1   520     up 9.24.104.1          0x10005AC85005 6611ral.itso.ral.ibm.com
     2  1070     up 9.24.104.14         0x400000032265 bnusbaum.itso.ral.ibm.com
     3   522     up 9.24.104.15         0x10005AC95049 barry.itso.ral.ibm.com
     4  1736     up 9.24.104.18         0x400000032262 eamon.itso.ral.ibm.com
     5  1646     up 9.24.104.19         0x400000033334 ssbreese.itso.ral.ibm.com
     6   524     up 9.24.104.23         0x10005A4F58CE rs60003.itso.ral.ibm.com
     7  1016     up 9.24.104.24         0x0080B21000B5 tridnxn2.itso.ral.ibm.com
     8  1022     up 9.24.104.25         0x10005AC95035 rs60005.itso.ral.ibm.com
     9  1706     up 9.24.104.26         0x10005AA8D769 rs60001.itso.ral.ibm.com
    10  1035     up 9.24.104.27         0x10005AC93F63 rs60004.itso.ral.ibm.com
    11   515     up 9.24.104.28         0x10005AA87023 rs60002.itso.ral.ibm.com
    12  1073     up 9.24.104.29         0x400000033306 ba33306.itso.ral.ibm.com
    13  1075     up 9.24.104.30         0x10005AA8B5EA rs60008.itso.ral.ibm.com
    14  7993     up 9.24.104.37         0x400000033352 p75itso.itso.ral.ibm.com
    15  1042     up 9.24.104.38         0x400000032232 lannetv.itso.ral.ibm.com
    16  1080   down 9.24.104.39         0x400000032249 bjohnson.itso.ral.ibm.com
    17  1082     up 9.24.104.40         0x400000032240 mcgregor.itso.ral.ibm.com
    18  2514   down 9.24.104.41         0x10005AACEEB2 itsoxst41.itso.ral.ibm.com
    19  1808     up 9.24.104.42         0x10005AEC2933 itsoxst42.itso.ral.ibm.com
    20  4209     up 9.24.104.45         0x10005AE8770A itsoxst45.itso.ral.ibm.com
    21  1044   down 9.24.104.47         0x10005AE88A04 itsoxst47.itso.ral.ibm.com
    22  1084     up 9.24.104.51         0x400000033324 itso51.itso.ral.ibm.com
    23  1086     up 9.24.104.54         0x400000033322 aixagent1.itso.ral.ibm.com
    24  1046     up 9.24.104.55         0x400000033342 aixagent2.itso.ral.ibm.com
    25  8525     up 9.24.104.58         0x400031740004 3174tcp1.itso.ral.ibm.com
    26  1048     up 9.24.104.70         0x10005AC92CEB 9.24.104.70
    27  1088   down 9.24.104.72         0x10005ACC5FD5 9.24.104.72
    28  1676     up 9.24.104.73         0x400000032293 9.24.104.73
    29  1037     up 9.24.104.74         0x400050013172 mvs18.itso.ral.ibm.com
    30  1090     up 9.24.104.76         0x10005AC92031 rs60007.itso.ral.ibm.com
    31  1092     up 9.24.104.78         0x10005A2080FD gildas.itso.ral.ibm.com
================================================================================
OVw id = 1018 - Network name = 9.24.1
IP Address = 9.24.1.0        - Subnet mask = 255.255.255.0
--> Contents of Segment: 9.24.1.Segment1
--------------------------------------------------------------------------------
         ObjID  Stat IP Address           Phys. Address  Node
--------------------------------------------------------------------------------
     1  1844     up 9.24.1.1            0x10005AC81099 6611slk.sl.dfw.ibm.com
     2  1019     up 9.24.1.3            0x10005AC830C8 6611ral.itso.ral.ibm.com
     3  1840     up 9.24.1.4            0x10005A88B612 gburg.sl.dfw.ibm.com
     4  1842     up 9.24.1.7            0x10005AC8D0D1 6611gbg.sl.dfw.ibm.com
     5  1838     up 9.24.1.9            0x10005AE83DF7 supername.sl.dfw.ibm.com
================================================================================

2 row(s) retrieved.
```

Figure 36. Result of wtqnetwork Command

3.7 Integrating SQL Queries Into the NetView for AIX GUI

The queries we have been using above will operate regardless of whether NetView for AIX is running (in fact, if the queries are issued on the server of a client/server database configuration, NetView for AIX does not have to be installed at all). In some cases it would be useful to have database information available at the NetView for AIX Graphical End-user Interface (GUI).

The method used for adding entries to the NetView for AIX menus, is to define the entry in a Registration File. Registration files are fully documented in *NetView for AIX Programmer's Guide,* SC31-6238. We produced a registration file to allow the user to execute the two sample programs `wtqnode` and `wtqnetwork` from a menu bar option when they have selected a node or network with the mouse.

The registration file we used is shown in Figure 37 on page 59. This adds a new option to the NetView for AIX menu bar called ″SQL″. The two entries added by the registration file use the NetView for AIX xnmappmon command to display the results of wtqnode and wtqnetwork in a scrollable Motif window.

The result of executing the wtqnode command from the SQL menu is shown in Figure 38 on page 60. Note that it is possible to make this action the default when you double-click a node with the mouse. This would produce a behavior similar to that of NetView for AIX Version 1, where summary details of the node are displayed instead of a submap containing IP interface symbols. You can make a node behave in this way by using the following procedure:

1. Select the node using the right mouse button (brings up a context menu).
2. Navigate the menus: Edit->Modify/Describe->Symbol
3. Change behavior to ″Execute″ instead of ″Explode″.
4. Select your application (in our case, ″SQL Queries : node″).
5. Select the node itself as target object.
6. Press **OK**.

```
/*
        Registration file for SQL database samples wtqnode
        and wtqnetwork
*/

Application ″SQL queries″ {

        MenuBar ″SQL″ _S
        {
                ″Contents of a node″     f.action   ″node″;
                ″Contents of a network″  f.action   ″network″;
        }
        Action ″node″
        {
             SelectionRule (isNode);
             MinSelected 1;
             MaxSelected 1;
        Command ″xnmappmon -commandTitle \″Node Detail\″
                 -geometry 625x540
                 -cmd /u/raleigh/SQL/informix/wtqnode \″$OVwSelection1\″″;
        }
        Action ″network″
        {
             SelectionRule (isNetwork);
             MinSelected 1;
             MaxSelected 1;
        Command ″xnmappmon -commandTitle \″Network Node List\″
                 -geometry 700x800
                 -cmd /u/raleigh/SQL/informix/wtqnetwork \″$OVwSelection1\″″;
        }
}
```

Figure 37. Registration File for SQL Query Samples

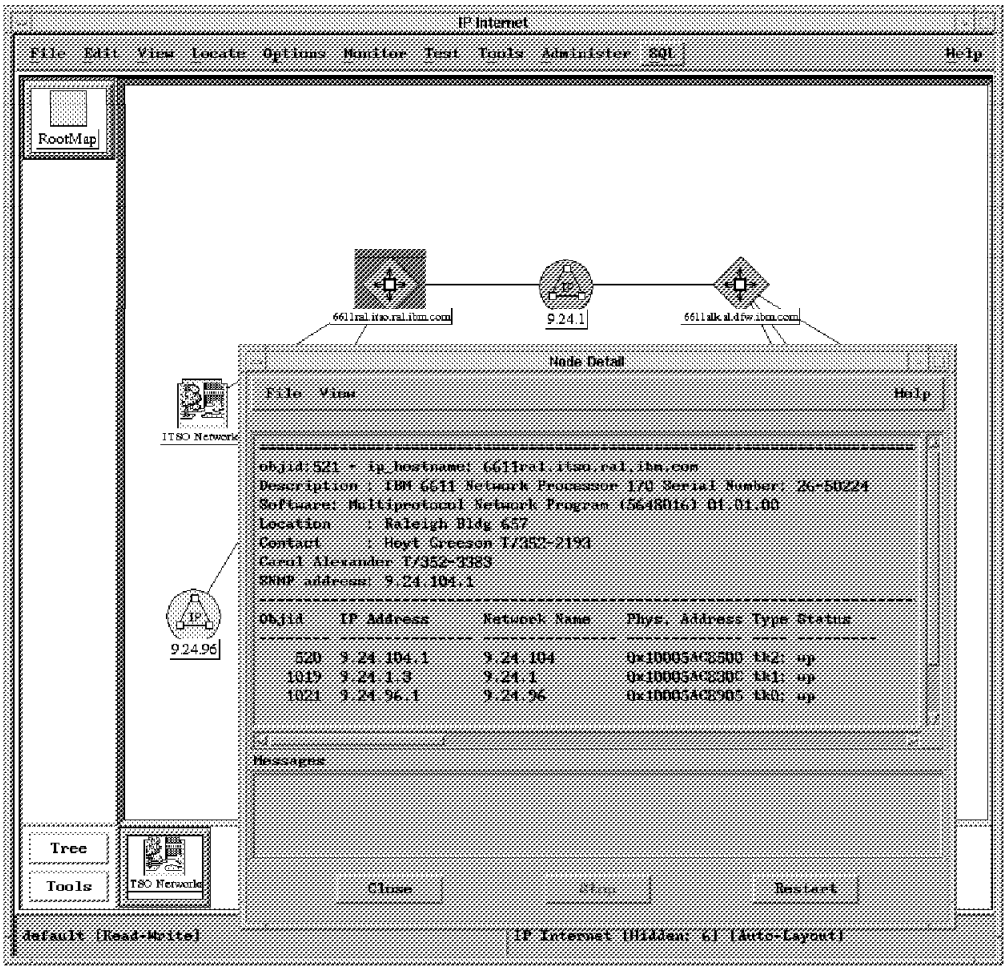


Figure 38. Executing wtqnode from a NetView for AIX Menu

3.8 trapdlog SQL Table

If you take the option to convert event log (`trapd.log`) data to SQL format, it is stored in the same database as defined for IP topology (above). The SQL table that contains the event information is called *trapdlog*.

In this section we will examine the trapdlog table and show an example of how the data collected in it may be used.

3.8.1 Structure of the trapdlog Table

Unlike the IP topology information, which is contained in many inter-related tables, trapdlog is a single table format. The columns correspond directly with the fields in the `trapd.log` flat file format (see 3.1.3, “The trapd.log File” on page 34).

Table 6. trapdlog SQL Table	
Database	**Description**
epochtime	Number of seconds since 01/01/70 (midnight)
trapcategory	Type of the trap • 0 = threshold events • 1 = network topology • 2 = error events • 3 = status events • 4 = node configuration events • 5 = application alerts • 6 = events displayed on all category windows • 7 = events logged but not displayed • 8 = map events, not displayed or logged
createtime	Represents the time the trap was created
ip_hostname	IP_hostname for the trap
trapsource	Represents the source of the trap • **a** = application • **A** = agent • **C** = Xnmcollect • **d** = demo or loadhosts • **D** = datacollector • **E** = Xnmevents • **G** = ignore • **I** = Ipmap_sa • **L** = loadmib • **M** = map or topmd • **n** = netmon related • **N** = netmon • **O** = Osi_sa • **P** = Nonip • **r** = Tralert • **s** = Spappl • **S** = security agent • **t** = Xnmtrap • **T** = trapd • **V** = vendor
description	Represents the description of the traps

3.8.2 Managing the trapdlog SQL Table

The process trapd, which received all traps to be processed by NetView for AIX, writes to the flat file /usr/OV/log/trapd.log even if the SQL database support is installed.

NetView for AIX provides utilities to copy the contents of the log file into the SQL trapdlog table, and to manage the table once it is loaded. The commands are:

traptosql This reads trapd.log and writes it to the trapdlog SQL table. You can optionally specify things such as which RDBMS to use, and where to read the log data from. The default is to use the RDBMS that you set up as default in SMIT (3.3.3, “Specifying Default RDBMS System” on page 43), and to use /usr/OV/log/trapd.log as the source.

trapdeletesql This command allows you to delete records from the trapdlog table based on criteria that you supply. You can specify dates, trap type, or host name to control the deletion.

trapsql We have met this command before (3.3.5.2, “Creating trapdlog SQL Table” on page 46) when creating the trapdlog table. It may also be used to clear and reinitialize the table.

It is important to put controls in place to manage the event log and the trapdlog SQL table. If you do not remove old records, the files will grow without limit.

We created a shell script, `trapdhousekeep` to help with this process. It is listed in Figure 39 on page 63. It performs three operations:

1. Records older than 14 days are deleted from the trapdlog SQL table
2. The current `trapd.log` is loaded into SQL
3. `trapd.log` is deleted

```
#!/bin/ksh

# number of days in each month

integer daysinmonth
set -A daysinmonth 31 31 28 31 30 31 30 31 31 30 31 30

# First work out the date two weeks ago (forget about
# leap years and the millenium!)

dd=`date +%e`
mm=`date +%m`
yy=`date +%y`

if (( dd < 15 ))
then
   ((mm = mm - 1))
   ((dd = dd + daysinmonth[mm]))
   if (( mm == 0 ))
   then
      mm=12
      ((yy = yy - 1))
   fi
fi

((dd = dd - 14))

# We may have knocked off leading zeroes - replace them

if [[ ${#mm} = 1 ]]
then
   mm=0$mm
fi
if [[ ${#dd} = 1 ]]
then
   dd=0$dd
fi
datetime="0000"
datetime=$dd$mm$yy$datetime

# Now delete all records over 14 days old
/usr/OV/bin/trapdeletesql -vT $datetime

# Move all records in trapd.log into SQL
/usr/OV/bin/traptosql -v

# Finally delete trapd.log
if (( $? == 0 ))
then
   cat /dev/null > /usr/OV/log/trapd.log
else
   print "traptosql failed - trapd.log not deleted"
fi
```

Figure 39. trapdhousekeep Shell Script

We want this housekeeping to be an automatic process, so it is a good idea to set up a crontab entry to perform it. We added the following crontab entry to cause the housekeeping to take place at 1 a.m. daily:

```
0 1 * * * /u/raleigh/scripts/trapdhousekeep
```

You add a new crontab entry by using the crontab -e command to edit the crontab table.

3.9 Using the Information in the trapdlog SQL Table

There are three options available to you for extracting data from the trapdlog SQL table:

1. The NetView for AIX provided command trapquerysql
2. General SQL SELECTs
3. Programs with embedded SQL

We will look at examples of all three approaches.

3.9.1 Using the trapquerysql Command

trapquerysql allows you to extract event records based on a range of criteria, such as hostname, date/time and event type. As an example, here we request all traps relating to a specific node:

```
/ > trapquerysql -H rs60003.itso.ral.ibm.com
767385777   3   Tue Apr 26 14:42:57 1994   A   rs60003.itso.ral.ibm.com    IBM Agent Up with No Changes (warmStart Trap)
767390023   3   Tue Apr 26 15:53:43 1994   A   rs60003.itso.ral.ibm.com    trapgend agent up with possible changes (coldStart trap)
767390469   5   Tue Apr 26 16:01:09 1994   A   rs60003.itso.ral.ibm.com    Application error
767390469   5   Tue Apr 26 16:01:09 1994   A   rs60003.itso.ral.ibm.com    Description  : Daemon Down
767390469   5   Tue Apr 26 16:01:09 1994   A   rs60003.itso.ral.ibm.com    IP Address   : 9.24.104.21
767390469   5   Tue Apr 26 16:01:09 1994   A   rs60003.itso.ral.ibm.com    Program      : SNA
767390469   5   Tue Apr 26 16:01:09 1994   A   rs60003.itso.ral.ibm.com    Location     : ITSC Raleigh
767390469   5   Tue Apr 26 16:01:09 1994   A   rs60003.itso.ral.ibm.com    Contact      : 919-301-1234
767390481   5   Tue Apr 26 16:01:21 1994   A   rs60003.itso.ral.ibm.com    Application error cleared
767390481   5   Tue Apr 26 16:01:21 1994   A   rs60003.itso.ral.ibm.com    Description  : Daemon Restored
767390481   5   Tue Apr 26 16:01:21 1994   A   rs60003.itso.ral.ibm.com    IP Address   : 9.24.104.21
767390481   5   Tue Apr 26 16:01:21 1994   A   rs60003.itso.ral.ibm.com    Program      : SNA
767390481   5   Tue Apr 26 16:01:21 1994   A   rs60003.itso.ral.ibm.com    Location     : ITSC Raleigh
767390481   5   Tue Apr 26 16:01:21 1994   A   rs60003.itso.ral.ibm.com    Contact      : 919-301-1234
```

Figure 40. Result of trapquerysql Command

3.9.2 Using SQL to Extract trapdlog Data

The trapquerysql command provides a limited amount of capability for extracting information from the trapdlog SQL table. By using simple SQL queries we can expand on this considerably.

For example, we may be interested in the availability of nodes in the network. NetView for AIX writes an event each time netmon status polling detects a node going up or down. By performing a simple SQL query we can extract these events and organize them.

The query we issued is shown below. It selects all ″Node Up″ and ″Node Down″ events, and then orders the results by hostname and time.

```
select ip_hostname,trap_create_time,description
from   trapdlog
where
       description like '%Node Up%'
or
       description like '%Node Down%'
order by ip_hostname,trap_create_time
;
```

Figure 41. SQL Select Command for trapdlog

The result of running this command is a report like that shown below:

```
trap_create_time  1994-04-26 15:52:52
ip_hostname       9.24.104.72
description       Node Up.

trap_create_time  1994-04-26 15:06:49
ip_hostname       9.24.105.2
description       Node Down.

trap_create_time  1994-04-26 15:08:51
ip_hostname       9.24.105.2
description       Node Up.

trap_create_time  1994-04-26 15:00:58
ip_hostname       alfred.itso.ral.ibm.com
description       Node Down.

trap_create_time  1994-04-26 15:05:58
ip_hostname       alfred.itso.ral.ibm.com
description       Node Up.

trap_create_time  1994-04-26 14:47:02
ip_hostname       joao.itso.ral.ibm.com
description       Node Up.

trap_create_time  1994-04-26 15:05:19
ip_hostname       joost.itso.ral.ibm.com
description       Node Down.

trap_create_time  1994-04-26 15:10:19
ip_hostname       joost.itso.ral.ibm.com
description       Node Up.
```

3.9.3 Using Embedded SQL with trapdlog

The report shown above for node up/down events has selected the data that we want, and gone some way towards organizing it. However, as with the IP topology examples, we would like a more sophisticated format. To achieve this we need to write a simple program.

The program sample `wttraplog` issues a modified version of the above SQL query, and then processes the output using the SQL cursor function, to match up Node Down/Up pairs. The source listing for the program is in Figure 194 on page 281. Sample output of the program is shown below.

```
================================================================================
Node name                          Down from:          to:                 Seconds
---------------------------------- ------------------- ------------------- -------

 9.24.105.2                        1994-04-26 15:06:49 1994-04-26 15:08:51 122

 alfred.itso.ral.ibm.com           1994-04-26 15:00:58 1994-04-26 15:05:58 300

 joost.itso.ral.ibm.com            1994-04-26 15:05:19 1994-04-26 15:10:19 300
================================================================================
```

Figure 42. Output from wttraplog Sample Program

3.9.4 Handling Multi-Line Events

The rows written in the trapdlog table have a one-to-one relationship with the records in the trapd.log flat file. The description that appears in the file is controlled by the trapd.conf file, which may be modified using the NetView for AIX Options->Event Configuration->SNMP Trap Configuration menu item.

It is possible to specify newline characters (″\n″) within the descriptive text, which improves the format of the event as it appears on the NetView for AIX event cards, for example, as shown in Figure 43. However, using the \n format causes multiple lines to be logged in the file /usr/OV/log/trapd.log (see Figure 44). Because of the one-to-one relationship between the log file and the rows in the SQL table, multiple trapdlog rows represent one original event.

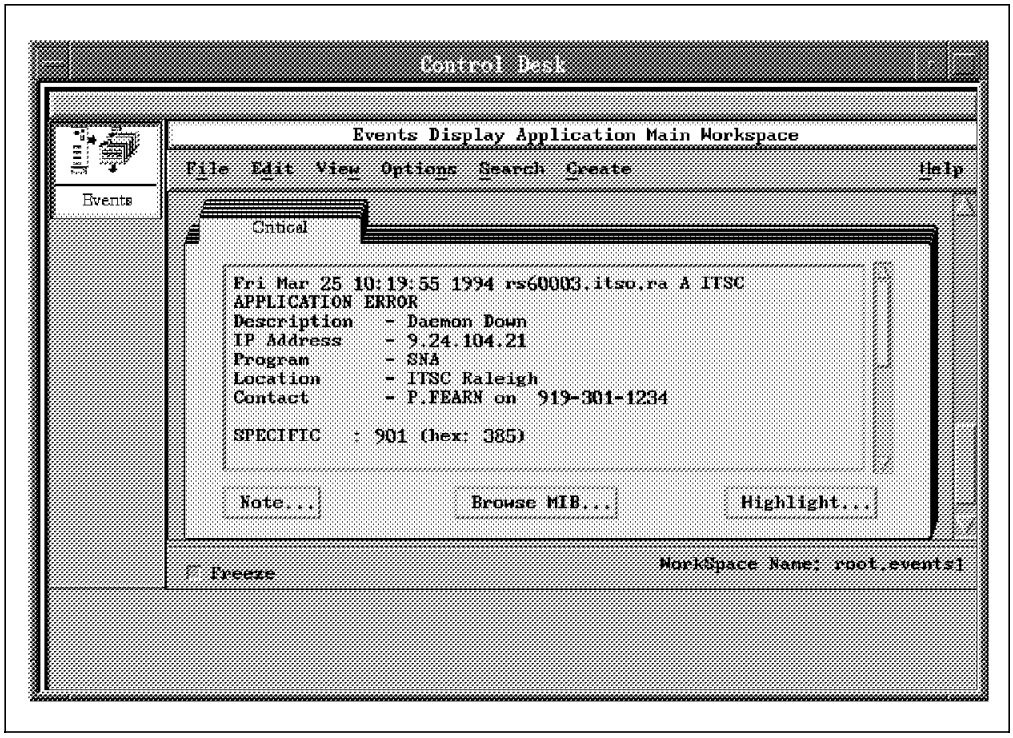


Figure 43. Application Trap

```
765577041  3 Tue Apr 05 16:17:21 1994 rs60002.itso.ibm.com A ITSC APPLICATION ERROR
765577041  3 Tue Apr 05 16:17:21      rs60002.itso.ibm.com A ITSC Description - Daemon Down
765577041  3 Tue Apr 05 16:17:21      rs60002.itso.ibm.com A Ip Address - 9.24.104.21
765577041  3 Tue Apr 05 16:17:21      rs60002.itso.ibm.com A Program - SNA
765577041  3 Tue Apr 05 16:17:21      rs60002.itso.ibm.com A Machine Location - ITSC Raleigh
765577041  3 Tue Apr 05 16:17:21      rs60002.itso.ibm.com A Contact - 919-301-1234
```

Figure 44. Contents of /usr/OV/log/trapd.log

Usually the fact that a problem is represented by multiple lines does not matter, but sometimes it does. For example, in the case shown above it may be that the ″Machine Location″ is a critical piece of information.

One way to handle the situation, when using SQL SELECT to find events, is to additionally select all events from the same node at the same time. This is not 100% guaranteed, however, since *epochtime* is only accurate to the nearest second.

An alternative approach is to preprocess the log file before loading it into SQL, concatenating descriptive text for multi-line events. The sample shell script wttrapconv (Figure 45 on page 67) does this. Note that as the maximum size of the description column in trapdlog is 256 bytes, the description may sometimes be truncated.

```
#!/bin/ksh

year=`date +%Y`
oldyear=year-1
infile="/usr/OV/log/trapd.log"
outfile="/tmp/trapd.log"
count_in=0
count_out=0

while read line
do
    ((count_in = count_in+1))
    set $line

    # First line of multiline has year in date - others do not
    if [[ $7 != $year ]] && [[ $7 != $oldyear ]]
       then
       line=`echo $line | cut -c59-`
       len1=${#line}
       len2=${#longline}
       longline="$longline / $line"
    else
       if (( count_out > 0 ))
       then
           print $longline >> $outfile
       fi
       longline=$line
       ((count_out = count_out + 1))
    fi
done < $infile

print $longline >> $outfile
print "trapd.log lines in: $count_in  trapd.log lines out: $count_out"

traptosql -vI $outfile
rm $outfile
```

Figure 45. wttrapconv Shell Script. This script builds long lines from multi-line trapd.log entries.

3.10 snmpCollect SQL Table

The snmpCollect SQL table, *coldata*, is very similar to trapdlog in structure and usage. The snmpCollect daemon polls for MIB data and stores it in its own proprietary format (see 3.1.1, “SnmpCollect” on page 33). It then has to be converted using the nvColToSQL command into the coldata table format.

The data being collected may be either raw MIB data or MIB expressions (that is, arithmetical combinations of MIB values). For this reason there are two additional tables, *varinfo* and *expinfo* which summarize the types of record found in the coldata file.

3.10.1 Structure of the snmpCollect SQL Table

Table 7. snmpCollect SQL Tables

Database	Description
coldata	The coldata table stores collected data, including a date/time stamp for the time of each data collection and the time in seconds since 1/1/70 for each data collection. • varID variable object identifier (Dotted Decimal) • expName user-specified expression name (mibExpr.conf) • collectTime beginning of collection interval(datetime) • startTicks beginning of collection interval (seconds) • stopTicks end of collection interval (seconds) • hostName host name where data was collected • ipAddr IP address where data are collected • instance variable instance (can be IP address) • stringValue IP address where data was collected • floatValue numeric value of the variable
varinfo	The varinfo table stores MIB variable data,including MIB variable object identifiers, names, type, and collection units. • varID variable object identifier (Dotted Decimal) • varName concatenation of mnemonic names (as defined in the MIB source) • varUnits units in which snmpCollect stored the data • varType GAUGE COUNTER IPADDRESS TIMETICKS INTEGER
expinfo	The expinfo table stores MIB expression data, including the label for the expression and a string representation of the expression in algebraic form. • expLabel expression user defined label • expression expression in algebraic form • expName user-specified expression name (mibExpr.conf) • expDesc user-specified expression description • expUnits represents the units of the collected expression

3.10.2 Managing the snmpCollect Data

The data in the snmpCollect SQL tables can be managed in much the same way as the trapdlog data (3.8.2, “Managing the trapdlog SQL Table” on page 61). Also the same caveats apply about putting housekeeping in place to prevent the collection files and SQL tables from becoming too large. We did not produce a sample script for this maintenance, but the trapdhousekeep sample (Figure 39 on page 63) could be easily modified to do it.

The commands provided for SQL table maintenance are:

nvColToSQL This converts files from the snmpCollect database into SQL tables. You have to specify the names of the files that you wish to convert. This means that any automatic processes to invoke nvColToSQL will have to be updated each time you add or remove a data collection process.

nvDColData This deletes rows from the SQL tables. You can set criteria for the rows to be deleted based on date, hostname and data type.

3.11 Using the Information in the snmpCollect SQL Tables

There are three options available to you for extracting data from the coldata, vardata, and expdata tables:

1. The NetView for AIX provided commands
2. General SQL SELECTs
3. Programs with embedded SQL

We will look at examples of the first two approaches.

3.11.1 Built-in Query Commands

There are three commands provided with NetView for AIX:

nvHostSumCol Prints a summary of the hosts for which data is in the database, and the number of records of each variable type.

nvDataSumCol Prints a summary of the data types (MIB variables and expressions) stored in the database, and the hosts to which they apply.

nvQColData Prints rows from the coldata table. You can specify arguments to restrict the dta extracted by date, hostname or data type.

First, an example of the nvHostSumCol command:

```
/ > nvHostSumCol
Variable Summary:

              Host                          VarID               Instance      Count
================================= ======================== ============== ======
rs6000.itso.ral.ibm.com           .1.3.6.1.2.1.2.2.1.10                  2     100
6611slk.sl.dfw.ibm.com            .1.3.6.1.4.1.2.6.2.4.1.1.2             1     320
6611ral.itso.ral.ibm.com          .1.3.6.1.4.1.2.6.2.4.1.1.2             1     332

Expression Summary:

              Host                          ExpName             Instance      Count
================================= ======================== ============== ======
No records found for MIB Expressions

nvHostSumCol completed successfully
```

Figure 46. nvHostSumCol Output

We see that the MIB variable IDs are expressed in "dotted decimal" form. If we want to see a more descriptive identifier we could issue the nvDataSumCol -V command, or use the NetView for AIX MIB Browser to display the textual description from the MIB source. In fact, the variables here are interface utilization (octets inbound) and 6611 Processor Utilization.

If we want to see some detail of the 6611 utilization figures we could use nvQColData:

```
/ > nvQColData -V .1.3.6.1.4.1.2.6.2.4.1.1.2

        VarID:          .1.3.6.1.4.1.2.6.2.4.1.1.2
        Instance:       1
        CollectTime:    1994-04-21 08:36:44
        StartTicks:     766931803
        StopTicks:      766931804
        HostName:       6611ral.itso.ral.ibm.com
        IpAddr:         9.24.104.1
        StringValue:    <null>
        FloatValue:     7

        VarID:          .1.3.6.1.4.1.2.6.2.4.1.1.2
        Instance:       1
        CollectTime:    1994-04-21 08:36:44
        StartTicks:     766931803
        StopTicks:      766931804
        HostName:       6611slk.sl.dfw.ibm.com
        IpAddr:         9.19.143.10
        StringValue:    <null>
        FloatValue:     2

        VarID:          .1.3.6.1.4.1.2.6.2.4.1.1.2
        Instance:       1
        CollectTime:    1994-04-21 08:41:42
        StartTicks:     766931804
        StopTicks:      766932102
        HostName:       6611ral.itso.ral.ibm.com
        IpAddr:         9.24.104.1
        StringValue:    <null>
        FloatValue:     2

 . . . . .
(many other entries)
```

Figure 47. nvQColData Command Example

3.11.2 Using SQL Select Commands with coldata

The above command has shown us all the 6611 utilization records that we have collected. The information we are more interested in, however, is ″which of our 6611s are experiencing high processor utilization values, and when″.

We can write a simple SQL query to discover this:

```
select
hostName, CollectTime, floatValue
from coldata
where     floatValue > 60
and       varID = '.1.3.6.1.4.1.2.6.2.4.1.1.2'
order by floatValue desc
;
```

Figure 48. Sample snmpCollect SQL Query

This query prints all instances where the 6611 utilization was over 60%, sorted in descending order.

3.12 Performance Considerations

Inevitably, using an RDBMS to store database information in place of direct file access will require additional processor cycles, memory, and I/O. In this project we did not attempt a thorough analysis of the impact using an RDBMS has on these performance indicators.

From the point of view of the user, these figures may not be significant anyway, since it is *perceived* performance that is critical. The trapdlog and snmpCollect data processes are essentially ″batch″ functions. The speed with which they complete is usually not of major importance to the user. For this reason we concentrate on the effect of using SQL for the ″live″ IP topology data.

In normal operation, there was *no* discernible difference between the SQL and non-SQL environments for the end user. We devised the following test as a way to measure one impact on the end user: We cleared the IP topology database and then restarted NetView for AIX from scratch, using non-SQL, local SQL and remote SQL configurations. In each case we measured the time taken (based on ″Node Added″ events being written to trapd.log) for all of the nodes in the local subnetwork (33 nodes in all) to be discovered. We repeated the test three times for each configuration to reduce random error and we performed the test without the GUI active, so that operator actions would not affect the timings. The results were as in Table 8.

Table 8. Network Discovery Performance Test

Config	Time	Comments
Non-SQL	57 seconds	
Local SQL	59 seconds	Very slightly slower than non-SQL. The difference would probably be more marked if the system were more heavily loaded.
Remote SQL	81 seconds	Hand-shaking between client and server probably account for the extra delay. We would expect this to be less sensitive to CPU load than the local SQL configuration, since the client code uses less system resource than the full database.

Again, it should be stressed that this test should not be extrapolated to other environments. All it proves is that, although there was no subjective difference between the configurations, small variations do exist. These variations may become important in a more heavily stressed system.

3.13 Extending SQL Support to the Object Database

As we have explained, of the ″openview″ databases, only IP topology is supported for SQL in NetView for AIX However there may be situations where it would be useful to combine information from the object database with the IP information. For example:

- Information about non-IP protocols
- Extra fields added to the object database for local use

During the project we developed a program `wtovwconv` to add an extra table to the SQL database and load it with information from the object database. The program source is listed in Figure 195 on page 283.

wtovwconv uses the OVw API to gain access to the object database (see Figure 21 on page 35 to see how the API is positioned within NetView for AIX). The program is in two parts. The first creates an SQL `create table` command to generate a table called *wtxxx* (where xxx is a name you supply). This table has a key of the object ID and contains fields matching each of the fields in the object database. The second part creates a sequence of SQL `insert` commands to add every network object into the table.

This two-part approach is necessary because the structure of the NetView for AIX object database is not fixed. That is, the fields in the database are defined by definitions in files in the *fields* directory: `/usr/OV/fields/C`.

wtovwconv does not directly perform the database load, but instead places the SQL statements into two `.sql` files. These may then be executed using whichever RDBMS is installed.

3.13.1 An Example of Using wtovwconv

In this example we show how an extra information field that we have added in the object database can be combined with data from IP topology in an SQL query. There are four steps to the process:

1. We add the field "LAN_connection" by placing a field registration file in the NetView for AIX fields directory. The file we used is shown below:

```
Field "LAN_connection" {
        Type    StringType;
        Flags   locate, general;
}
```

2. We need to give the new field a value for some of the nodes in the network. We could write a program to access the OVw API and update the field (*Examples Using NetView for AIX APIs* GG24-4059, includes sample code that can do this). However, as we made it a field of type *general* we can simply select Edit->Modify/Describe->Object from the NetView for AIX menu bar and type in the value we want.
3. We run the sample program `wtovwconv` to generate the two SQL command files as described above
4. We use the facilities provided by the RDBMS to invoke the command files, thereby creating and loading the database table.

Figure 49 on page 73 shows the result of running a modified version of the `wtqnode` program (see 3.6.1, "SQL Sample wtqnode" on page 55) with this locally-added information included. Figure 50 on page 73 shows the modified SQL `select` command that joins our new table with the interface table to generate the modified output.

```
================================================================================
objid:541 - ip_hostname: rs60005.itso.ral.ibm.com
Description : IBM RISC System/6000
Machine Type: 0x0101 Processor id: 000014723800
The Base Operating System AIX version: 03.02.0000.0000
TCPIP Applications version: 03.02.0000.0000
Location    : IBM B657, 4912 Green Road, Raleigh NC 27609
Contact     : Dave Shogren
SNMP address: 9.24.104.25
--------------------------------------------------------------------------------
Objid    IP Address      Network Name    Phys. Address Type Status LAN attachment
-------- --------------- --------------- ------------- ---- ------ -----------------
    540  9.24.104.25     9.24.104        0x10005AC9503 tr1;     up 8230 #2, Port 16
    580  9.67.32.84      9.67.32.64      0x08005A0D7FC en0;   down Not connected
    581  9.67.32.86      9.67.32.64      0x08005A0D7FC et0;   down Hub 9, Slot 5, Port3
    583  9.67.46.170     9.67.46.128     0x10005AB155D tr0;   down 8250 #1, Slot 2, Port 19
================================================================================

1 row(s) retrieved.
```

Figure 49. Modified wtqnode with Manually-Added Data from Object Database

```
select interfaceclass.ip_address,
       ip_network_name,
       snmp_ifphysaddr,
       snmp_ifdescr,
       interfaceclass.ip_status,
       lan_connection
from   interfaceclass,
       coupledwith,
       objecttable,
       networkclass,
       wtobj
where  interfaceclass.objid=$interface_id
and    coupledwith.objid2=interfaceclass.objid
and    coupledwith.objid1=objecttable.objid
and    objecttable.classid=1
and    wtobj.objid=interfaceclass.objid
;
```

Figure 50. SQL Select Command to Extract Additional Object Data. The name we gave the object database table in SQL was "wtobj" in this case.

Chapter 4. Event Configuration

This chapter provides a summary of NetView for AIX event configuration and shows examples of using NetView for AIX event configuration support.

NetView for AIX, including its event configuration, alert editor and RUNCMD support, together with the ability to enter this environment via either a trap or AIX V3 errlog entry, can be an effective part of an installation's problem solving and network or systems management solution. This is indicated in the following figure.

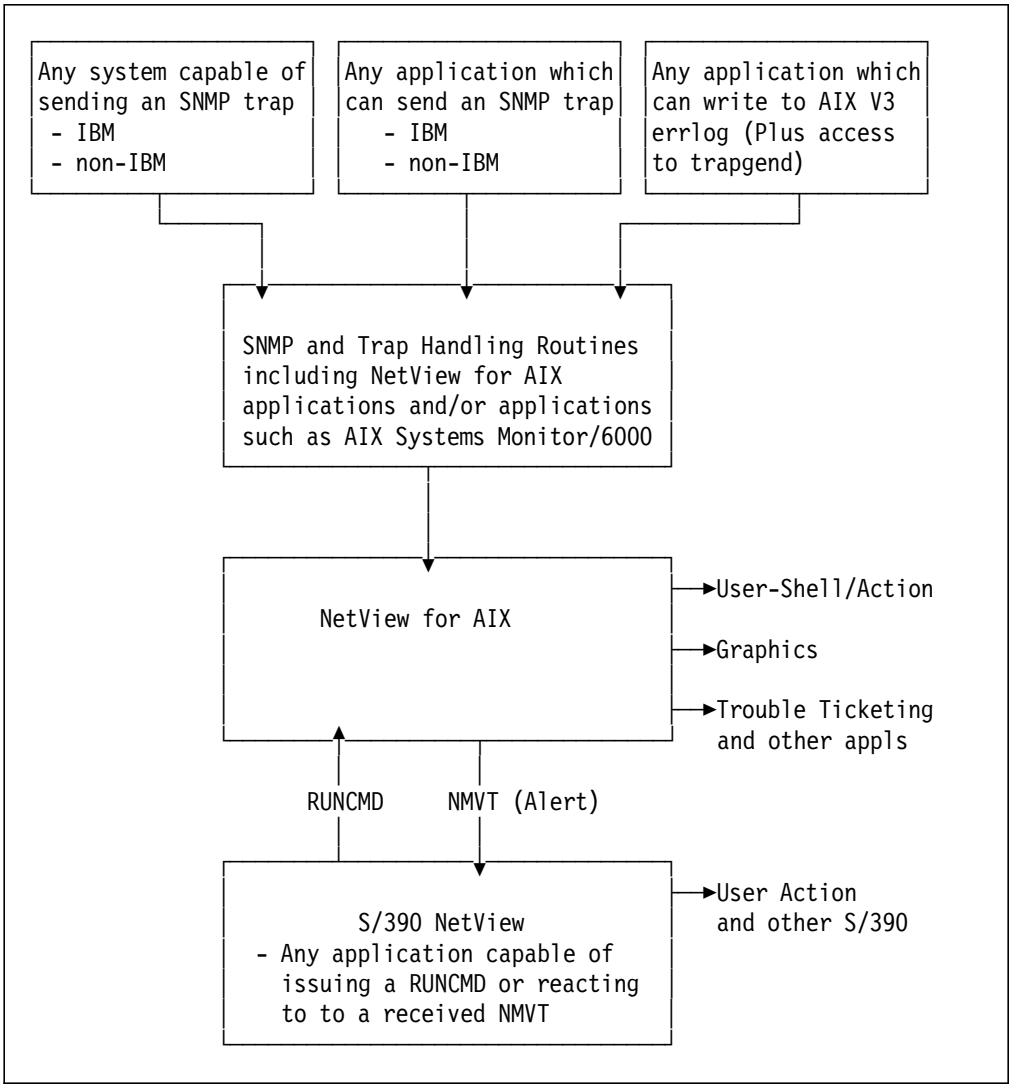


Figure 51. NetView for AIX Event Configuration Ins and Outs

4.1 Summary of AIX V3 Event, Trap and Alert Management

The terms ″trap″, ″event″ and ″alert″ have various definitions and are used in various ways. This chapter explains how these terms are used in this document.

4.2 NetView for AIX Events and Traps

There are two types of NetView for AIX events:

1. A map event is generated when a user or an application changes the status of the current open map.
2. A network event is sent by an agent providing information regarding a change in the network. In the SNMP environment these events are also called traps.

Simple Network Management Protocol (SNMP) defines six generic types of traps and also allows for the definition of enterprise-specific traps. The application events are treated as enterprise-specific traps.

NetView for AIX uses several daemons and other processes to manage the network. These processes will be synchronized with each other when NetView for AIX is initially started.

For the duration of this section the term event, will also refer to traps. When the events arrive, they will be formatted into an enterprise-specific format.

Events may be raised by SNMP functions, applications, NetView for AIX components, shell scripts, AIX operating system and other SNMP-managed devices.

4.3 NetView for AIX Event and Trap Daemons

The following sections outline the key NetView for AIX processes (daemons) for managing events and trap processing. Figure 52 on page 77 summarizes the daemons and shows how they communicate with each other.

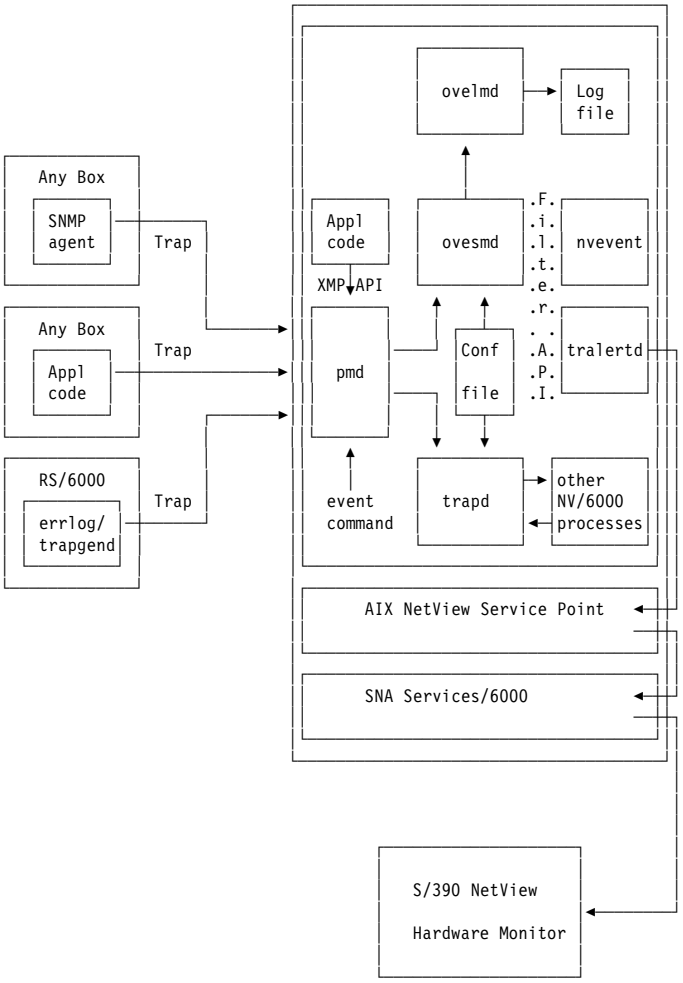


Figure 52. NetView for AIX V3 Daemons

4.3.1 NetView for AIX Daemons and Agents Raising Events

snmpd The snmpd daemon is a server that services requests for the SNMP protocol. It provides responses to SNMP requests from network monitor applications, processing requests and returning the results to the network monitor. It also sends trap notifications to all hosts configured to receive the traps. All actions are logged in the file `/usr/tmp/snmpd.log`.

smux subagents These subagents interface with the snmpd daemon using the SNMP multiplexor (SMUX) protocol. These applications register with the snmpd daemon to support Management Information Base (MIB) extensions, and reply to SNMP requests. They can also forward traps to the snmpd daemon, which will then forward them to the network monitor.

trapgend The trapgend daemon is an example of a SMUX subagent that converts AIX alertable errors into SNMP traps.

netmon The netmon daemon is responsible for discovering nodes and then regularly polling them for new status. This uses the ICMP echo request, but also will use SNMP GET requests if the device is running an SNMP agent. When changes happen in the network, netmon will generate an event, which it passes to trapd for processing.

snmpCollect The snmpCollect daemon polls systems for MIB values. Unlike the polling for status information, the snmpCollect daemon only polls for MIB data that is specified in the NetView for AIX Data collection configuration option. Events are raised when threshold values are exceeded.

4.3.2 NetView for AIX Daemons Acting on Events

pmd pmd (the ″postmaster″ daemon) allows processes to access all protocols (SNMP and CMOT) and manages message routing to managers and agents. All the traps raised pass through this daemon. The traps are then sent to the ovesmd and trapd daemons.

trapd The trapd daemon receives SNMP traps from agent nodes, and then sends them to the connected daemons. The trapd daemon logs all the SNMP traps in the default log file `/usr/OV/log/trapd.log`.

The trapd daemon also forwards SNMP traps to other applications that are directly connected to it (such as netmon).

ovactiond The ovactiond daemon is responsible for acting on any events sent to the NetView for AIX system whilst the NetView for AIX GUI is not active. This is a new daemon supplied with NetView for AIX V3R1.

ovesmd The Event Sieve Agent, ovesmd, receives events from the pmd daemon. The ovesmd daemon determines which application should receive the event. These applications can be user applications based on SNMP or XMP APIs or NetView for AIX services. For example, nvevent displays the events on the NetView for AIX GUI. ovesmd will also handle the filtering according to the filter rules defined in NetView for AIX.

ovelmd The event log agent, ovelmd, stores events in the log file `/usr/OV/log/ovevent.log`. This log file is used for the NetView for AIX event history display application.

tralertd The tralertd daemon converts traps into alerts. These are passed to it across the ovesmd filter interface. After conversion, they are forwarded to the NetView Host machine via the AIX NetView Service Point application.

spappld The spappld daemon receives RUNCMDs from NetView and then passes them to the service point application. It also provides services to return a reply to the host.

gtmd The gtmd daemon receives events generated from non-IP agents that use the generic topology MIB format. An example of this is LMU/2 interfacing with LMU/6000, which can send IPX information for a Novell NetWare network to NetView for AIX. The information is then formatted by the NetView for AIX xxmap application.

4.4 SNMP Configuration for AIX

Traps are messages formatted according to RFC 1215. The RFCs (″The Request For Comments″) document the defined standards of the Internet suite of protocols. The advantage in using traps is that NetView for AIX may manage heterogeneous networks that use SNMP protocols.

The AIX V3 SNMP Agent, snmpd, knows which management system to send the traps by use of a configuration file /etc/snmpd.conf.

The comments lines at the top of the /etc/snmpd.conf file explain in some detail the configuration lines required. A section of a working snmpd.conf file can be seen below.

/etc/snmpd.conf extract

```
logging  file=/usr/tmp/snmpd.log  enabled
logging  size=10000   level=3

community public  0.0.0.0     0.0.0.0         readOnly
community ITSO    127.0.0.1   255.255.255.255 readWrite 1.17.2
community ITSO    9.67.38.71  255.255.255.255 readWrite # rs60001
community ITSO    9.24.104.28 255.255.255.255 readWrite # rs60002
community ITSO    9.24.104.23 255.255.255.255 readWrite # rs60003
community ITSO    9.24.104.27 255.255.255.255 readWrite # rs60004
community ITSO    9.67.38.67  255.255.255.255 readWrite # rs60005
community ITSO    9.24.104.0  255.255.255.0   readWrite # any itsc

view            1.17.2          system enterprises view

# Avoid collisions with the machine list
# trap ITSO 127.0.0.1  1.3.1 fe # loopback
trap  ITSO rs60001  1.2.1 fe
trap  ITSO rs60005  1.2.5 fe

#snmpd  maxpacket=1024 querytimeout=120 smuxtimeout=60

smux  1.3.6.1.4.1.2.3.1.2.1.2  gated_password # gated
smux  1.3.6.1.4.1.4.3.1                        # unixd
smux  1.3.6.1.4.1.2.6.12       sm6000    # Systems Monitor/6000:  sysmond
smux  1.3.6.1.4.1.2.6.4.1      nv6000    # NetView for AIX: trapgend
```

The /etc/snmpd.conf file contains the following items:

1. SNMP log information
2. A list of valid community names
3. SNMP trap destinations
4. SMUX Agent MIB registration information

Table 9. /etc/snmpd.conf community Extract

1	2	3	4	5	6	7
community	ITSO	9.24.104.0	255.255.255.0	readWrite	iso.3	# ITSO

Notes:

1 Identifies this as a community entry in the /etc/snmp.conf file.

2 This is the community name of this network.

3 This is the desired network that will be allowed access to the MIBs controlled by this station.

4 This is the mask used to AND the IP address of the requesting IP host in order to validate the request.

5 This is the type of the MIB access that will be given if the ANDed result of the requesting IP host with 4 is equal to 3.

6 Signifies the view; that is, what section of the MIB tree can be accessed by the requester.

7 Comments follow the # sign.

When a host makes a request to access the MIB, then the IP address of the requesting host is ANDed with 4. If the result is equal to 3 then the access defined by 5 is granted to the requesting host. This assumes that the request was made using the 2 community name.

Table 10. /etc/snmpd.conf trap Extract

1	2	3	4	5
trap	ITSO	rs60002	1.2.1	fe

Notes:

1 Identifies this as a trap entry in the /etc/snmpd.conf file.

2 This is the community name for the destination machine.

3 This is the hostname of the machine the trap will be sent to.

4 This is a view name. This is not used in AIX V3.

5 This field is used for blocking traps, the Hex 'fe' is converted into binary, and each bit is associated with a trap type. (Coldstart, warmstart, linkdown, linkup, authentication, egpneighborloss and enterprise-specific). The value fe corresponds to "No blocking of traps" whereas X'be' (10111110 in binary) would block the warmstart trap.

Table 11. /etc/snmpd.conf SMUX Extract

1	2	3	4
smux	1.3.6.1.4.1.2.6.12	sm6000	# AIX Systems Monitor/6000 sysmond

Notes:

1 Identifies this as a smux entry in the /etc/snmpd.conf file.

2 This identifies the section of the MIB tree this agent is registered for.

3 This field is an ID tag for the agent. This is in effect a password linking this file with /etc/snmpd.peers for authentication.

4 Comment field.

4.5 NetView for AIX Events

The NetView for AIX application that displays events, using a graphical user interface, is called nvevent.

With Version 3 of NetView for AIX a number of changes have been made to this interface, including:

Dynamic Workspaces

These workspaces will be updated with new events as they arrive. The previous version of NetView for AIX only allowed one dynamic workspace; all others were static workspaces. The static workspaces are never updated when new events arrive. The dynamic workspaces can be used to select events from:

- A specific source
- A specific severity of event
- A specific category of event
- Filter control for the event

A number of dynamic workspaces can be opened at any one time. For example, one dynamic window could show the events with a severity of Critical, and another workspace could show events generated from a specific machine.

The Events application

A number of new options have been added to the pull-down menu bar contained in the Events window. Most of these new options will be discussed in the following sections of this chapter.

Event Card Layout

The screen that displays the events has changed to show more of the information contained on the cards.

The events application within NetView for AIX has a number of aspects:

1. When NetView for AIX initially starts, the events EUI application will, by default, be started and attached to the base of the main NetView for AIX screen. This area is called the Control Desk. This portion of the screen can be moved into a separate menu by pressing the middle mouse button while the pointer is on the Control Desk bar, and then using ″drag and drop″.
2. If NetView for AIX is started, and the Events window has been closed as could have been done via the panel in Figure 53 on page 82, then from the NetView for AIX pull-down menu select:

   Monitor -> Events -> Current Events

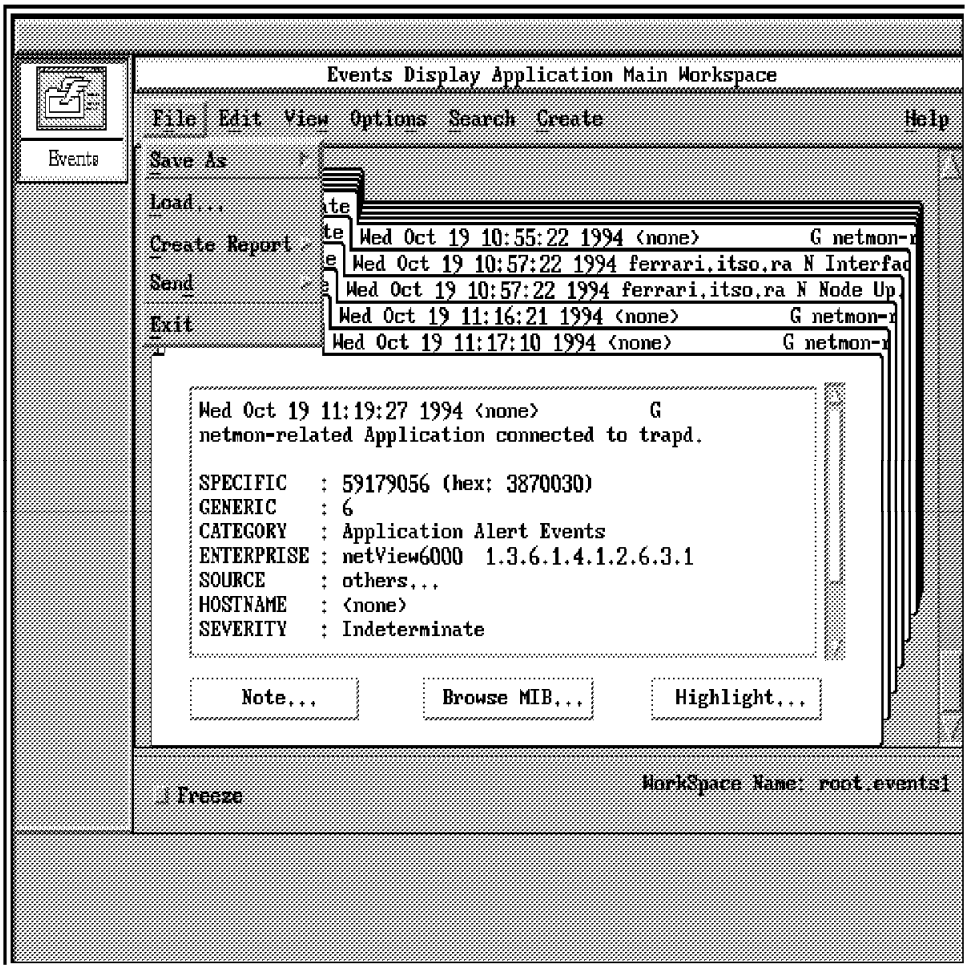


Figure 53. Events Window Showing Exit (Close Events) Option

Closing via *File/Exit* results in stopping the nvevents task associated with the particular Events Window.

The following is an example of this:

- First, one Map (operator) had one Events Window active:

```
[root@rs60003]/> ps -ef|grep nvevents
    root 50822 47502   0 11:19:27      -  0:13 /usr/OV/bin/nvevents
    root 51166 50121   2 14:46:02  pts/0  0:00 grep nvevents
```

- Then, a second Map opened an Events Window. This second Events Window could have been initiated by the first Map operator using approaches which will be discussed later in this chapter.

```
[root@rs60003]/> ps -ef|grep nvevents
    root 50822 47502   1 11:19:27      -  0:13 /usr/OV/bin/nvevents
    root 52392 51799   0 14:47:15      -  0:05 /usr/OV/bin/nvevents
    root 54324 50121   1 14:49:24  pts/0  0:00 grep nvevents
```

- Next, the first Map closed its Events Window.

```
[root@rs60003]/> ps -ef|grep nvevents
    root 50751 50121   2 14:50:21  pts/0  0:00 grep nvevents
    root 52392 51799   0 14:47:15      -  0:05 /usr/OV/bin/nvevents
```

- Followed by the first Map re-opening its Events Window.

```
[root@rs60003]/> ps -ef|grep nvevents
    root 50820 47502  20 14:51:38      -  0:05 /usr/OV/bin/nvevents
    root 52392 51799   0 14:47:15      -  0:05 /usr/OV/bin/nvevents
    root 54408 50121   1 14:51:57  pts/0  0:00 grep nvevents
```

3. Rather than using **Monitor -> Events -> Current Events**, from the AIX command line, the operator could type: /usr/OV/bin/nvevents &

4. Or, if the toolbox window is present, then two approaches could be used in conjunction with the **Events** icon.

   a. If the Events window is not active, drag and drop the *Events* icon from the toolbox icon into the Control Desk or to some open area in the display.

   b. Select (via left-button click) on a node in a submap and double click the toolbox *Events* icon to open an Events Window aimed at events for the selected node.

      Figure 54 on page 83 through Figure 57 on page 86 is an example of this.

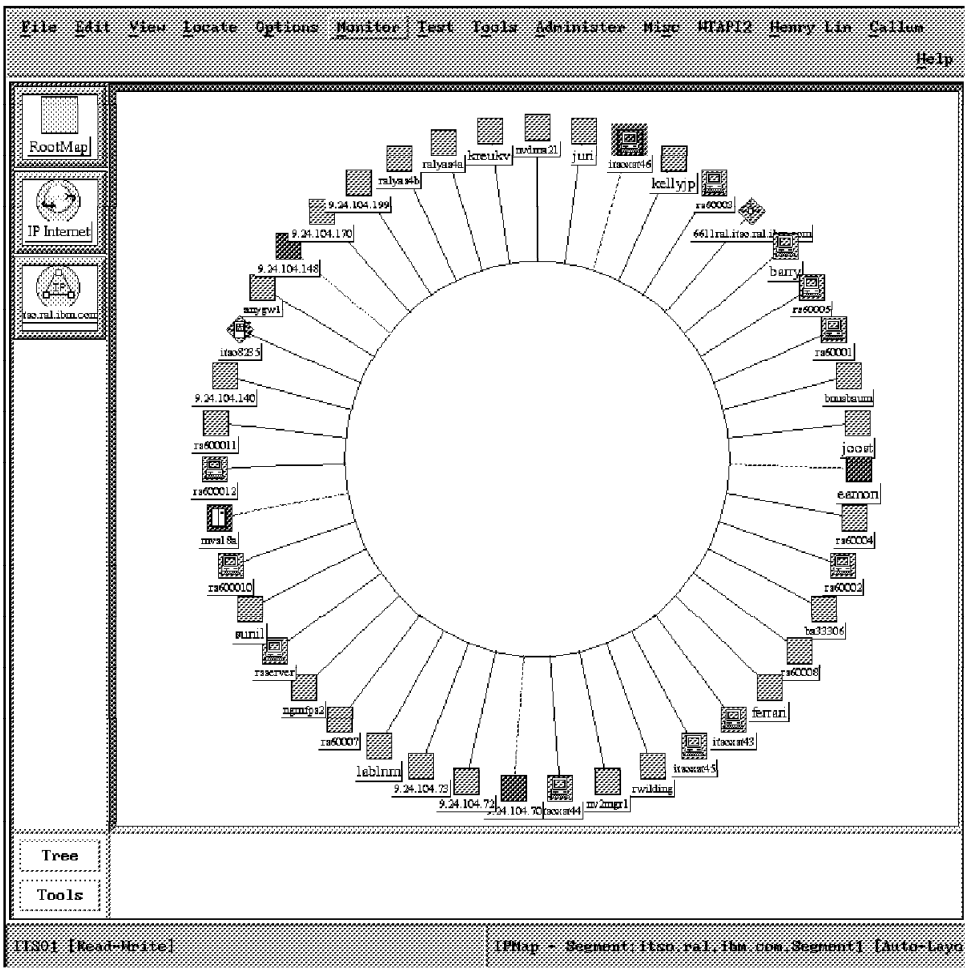


Figure 54. A Selected Node

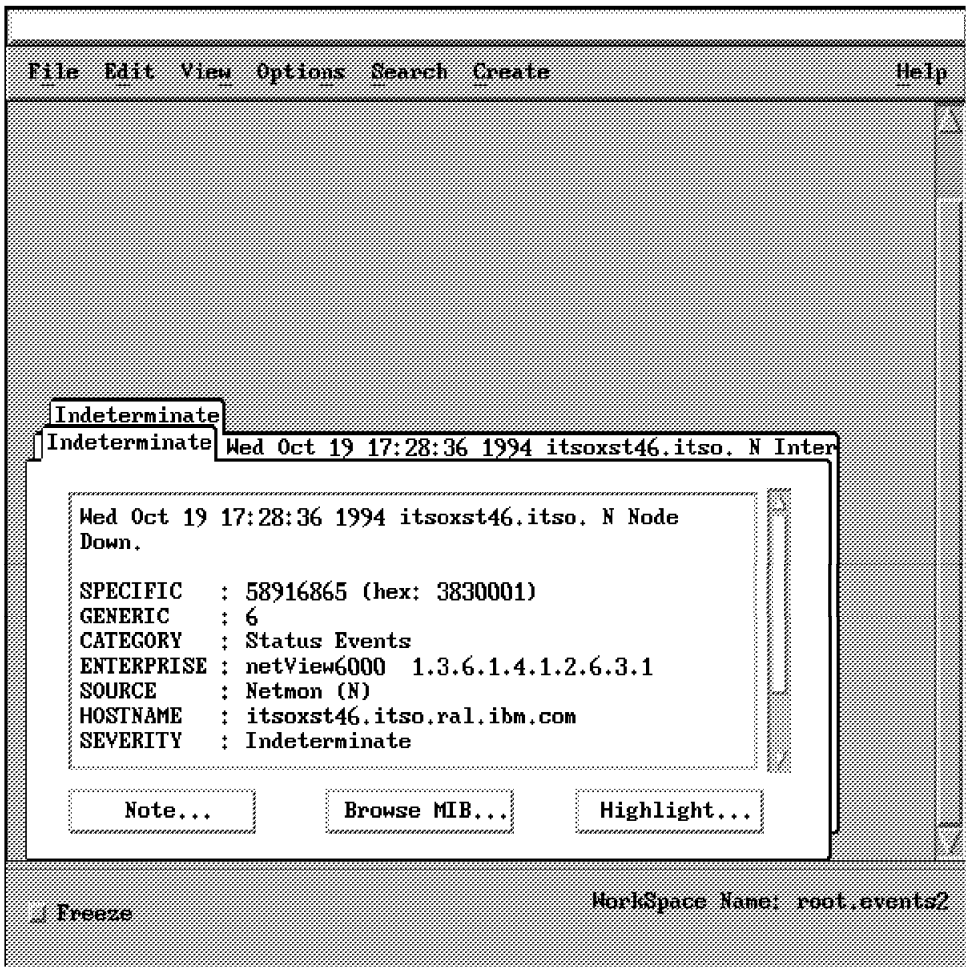


Figure 55. The Events Window (Main Events)

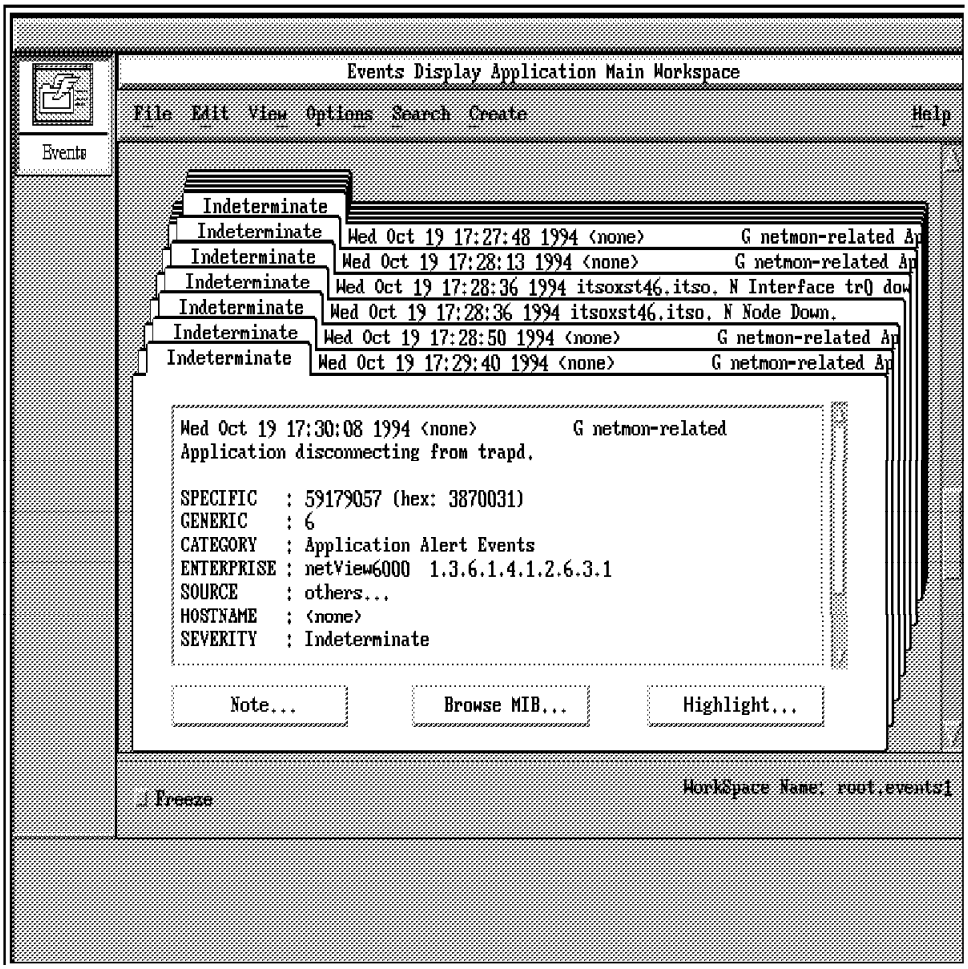


Figure 56. The Events Window (For Selected Node's Events)

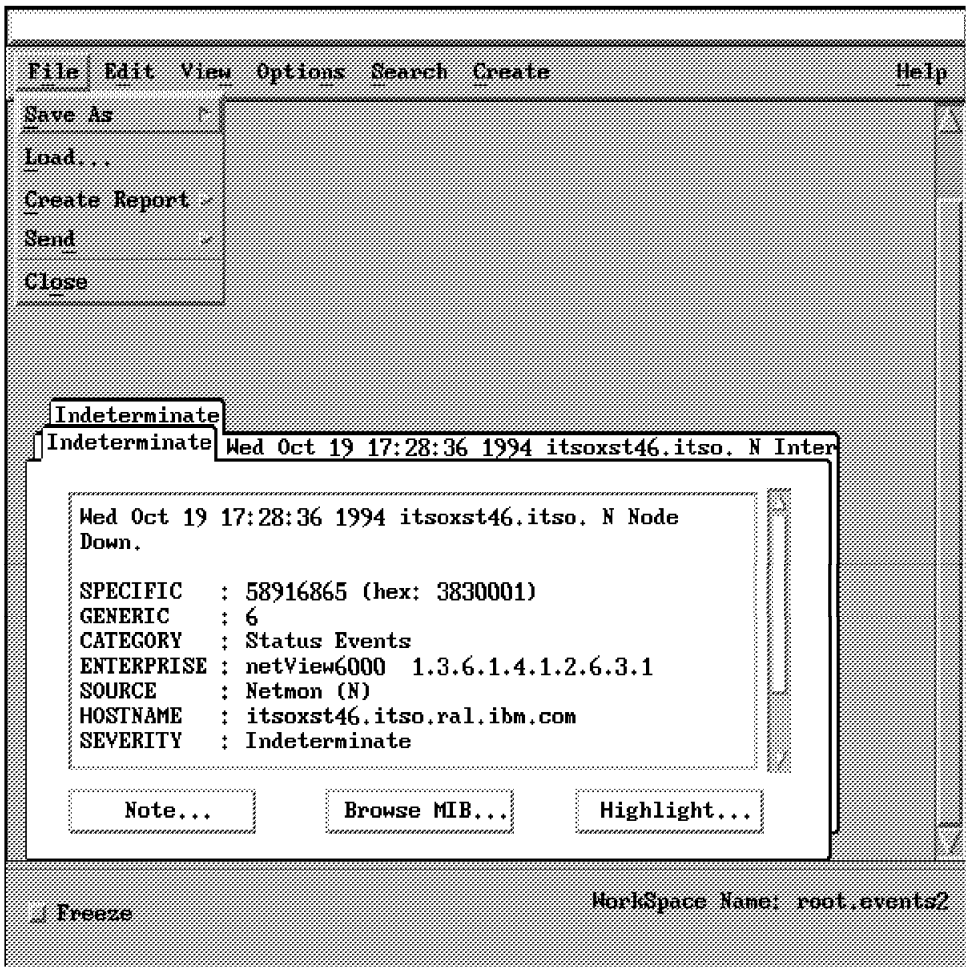


Figure 57. Option for Closing Selected Node's Events. Closing only affects this set of filtered events and does not stop nvevents as previously discussed in Figure 53 on page 82.

The steps above begin the nvevents application, which reads the events information from the /usr/OV/log/ovevent.log file.

Note:

If you do not want the Events window to appear when NetView for AIX is started, then do the following:

1. Edit the file /usr/OV/registration/C/ovsnmp/nvevents.
2. Amend the Command line from:
   - Command -Shared -Initial "${nvevents:-/usr/OV/bin/nvevents}";

     to:
   - Command -Shared "{$nvevents:-/usr/OV/bin/nvevents}";
3. Restart NetView for AIX.

To start the Events window simply drag Events from the Tools window, or optionally select **Monitor-> Events->Current Events...** from the NetView for AIX pull-down menu.

The Events window display can be modified to show additional information, including:

- The maximum number of events displayed to the user
- The maximum number of events to be loaded from ovevent.log
- The initial presentation of the events to the user (that is, card or list format)
- The name of the default filter file
- The colors and fonts on the cards

All of the above are controlled by parameters that you will find in the /usr/lpp/X11/lib/X11/app-defaults/Nvevents file.

This file should not be directly modified but should be copied to the users home directory and modified there. To modify the options for a particular user, do the following:

- Change directory to the user's $HOME directory.
- Append /usr/lpp/X11/lib/X11/app-defaults/Nvevents to the $HOME/.Xdefaults file. If $HOME/.Xdefaults does not exist then it can be created.
- Modify the options as required using an editor such as vi.
- Save the file.
- Restart NetView for AIX.

An example of this file is provided in Appendix F, "Nvevents X11 app-defaults File" on page 289.

4.5.1 The NetView for AIX Event Screen

Figure 58 on page 88 shows the NetView for AIX initial screen display, as started by the /usr/OV/bin/nv6000 script. The display shows the event configuration screen attached to the NetView for AIX's display of the current managed network. Throughout this document references will be made to the pull-down menu options located on the top half of this screen.

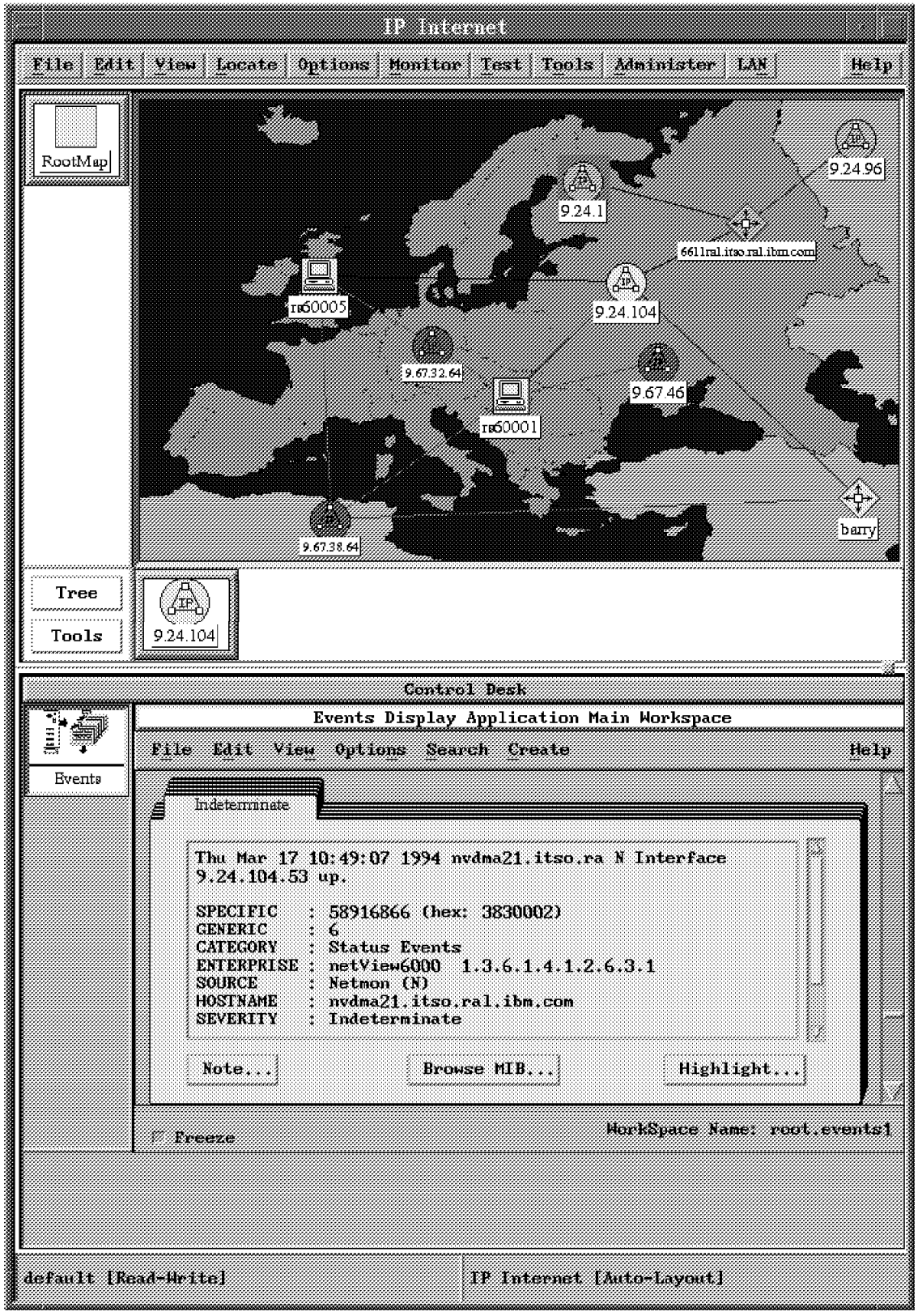


Figure 58. An Example of NetView for AIX Initial Screen Display

Figure 59 on page 89 shows a trap as displayed by NetView for AIX. This trap was generated by the netmon daemon and signifies a Node Up network event.

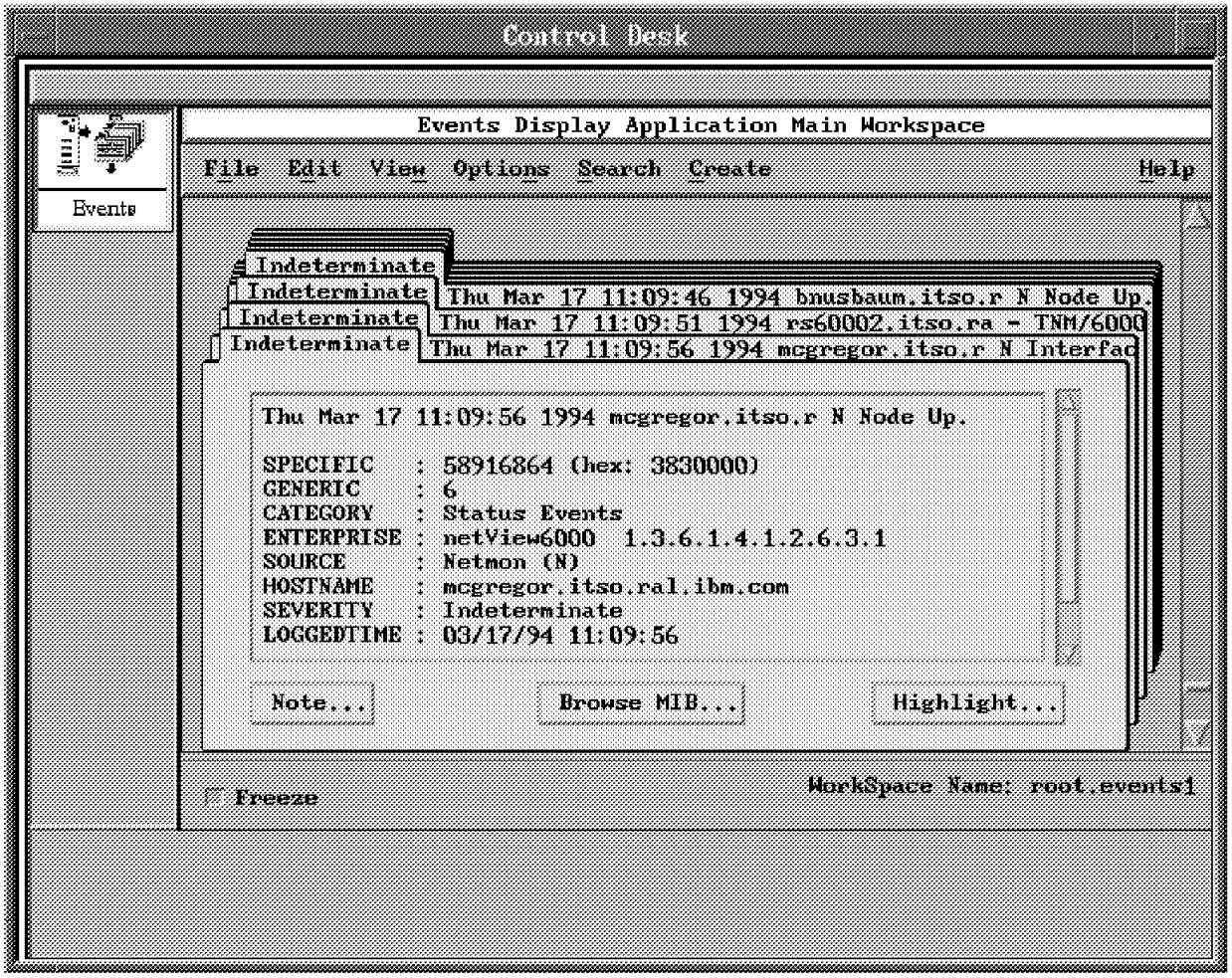


Figure 59. An Example of NetView for AIX Event Card

The display shows the default event information. This particular event is NetView for AIX specific. You can customize the display window to display the required information that relates more to the specific event being logged.

As seen in Figure 59, the Event Card window provides the following:

- The top left hand corner of the card signifies the severity of the event. The first line of the card shows the event summary.
- The three buttons located at the bottom of the event allow you to do the following:
  - Note... Allows notes to be recorded for a specific event.
  - Browse MIB... Access to the machine's MIB values.
  - Highlight... Highlight the symbols on the map representing the device that raised the trap.

Clicking on any of these buttons is a fast-track approach for making the above operator actions. Clicking on the Highlight button brings forward a submap containing the highlighted-device, clicking on Browse MIB brings forward the MIB browser and clicking on Note brings forward the Note Editor.

4.5.2 Event Card Information

The information contained within the event window main section includes:

SPECIFIC : 58916864 (hex: 3830000)
The enterprise-specific trap ID for the Node Up event.

GENERIC : 6
The generic trap ID. Generic 6 means the event is an enterprise-specific event.

CATEGORY : Status Event
Indicates that that there has been a change in the network.

ENTERPRISE : netView/6000 1.3.6.1.4.1.2.6.3.1
The enterprise-specific ID, which indicates the origin of the trap. In this case it indicates NetView for AIX, meaning that this is an internally-generated event.

SOURCE : Netmon
The event was raised by the netmon daemon.

HOSTNAME : mcgregor.itso.ral.ibm.com
The hostname affected by the event (for an event with an external origin, this would be the node from which the trap originated).

SEVERITY : Indeterminate
The severity type is indeterminate.

LOGGEDTIME : 03/17/94 11:09:56
The time the event was generated.

4.6 NetView for AIX Event Configuration

NetView for AIX has a number of events initially configured. These are IBM and vendor-specific.

When a new NetView for AIX application is installed (for example, AIX Systems Monitor/6000), then additional enterprises, with additional enterprise-specific events, are also configured. If the application at install time does *not* configure its events (traps) it is the user's responsibility to do so.

All the defined events for NetView for AIX, are located in the file `/usr/OV/conf/C/trapd.conf`.

The Event Configuration window (Figure 62 on page 95) can be displayed from the main NetView for AIX window, by selecting the following from the pull-down menu:

- **Options->Event Configuration->Trap Customization**
- Or by typing the command:

  ```
  usr/OV/bin/xnmtrap &
  ```

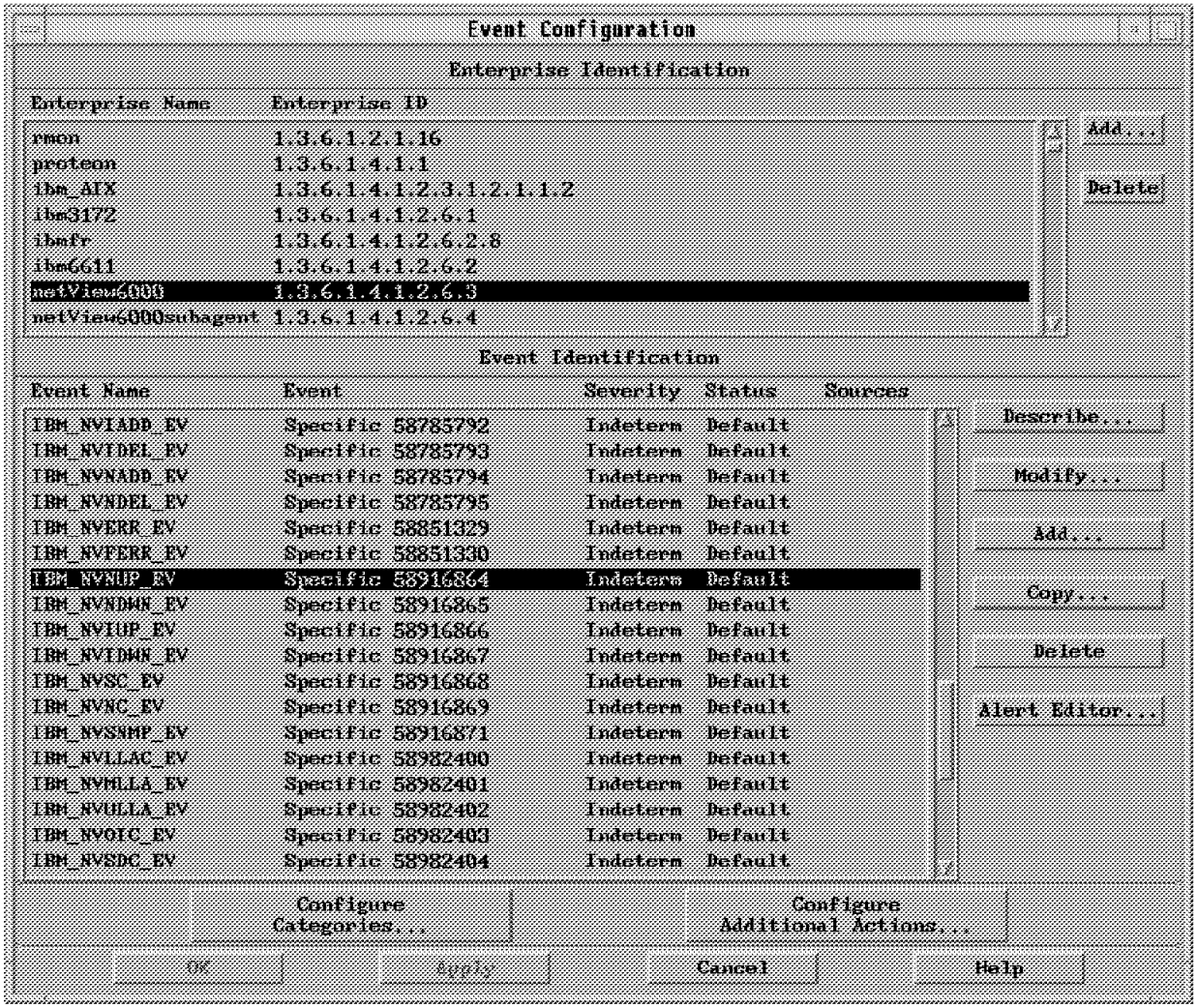


Figure 60. NetView for AIX Event Configuration Window

If you use the event configurator, you can browse the installed enterprise definitions and look at their associated specific events in the Event Identification window listing the Event Name, event number, severity, status and source.

In Figure 60, we see the definition of the Node Up event.

To list all the NetView for AIX events, type event -l (see Appendix E, “NetView for AIX Default Events” on page 287).

The output from this command shows all the event types, their event numbers, and associated descriptions.

There may be a requirement to add a new event to the associated enterprise. You may also modify existing events to display more meaningful messages on the event card.

To generate a Node Up event from the AIX command line, type:

```
event -l | grep "Node Up"
```

This will display the line:

```
NUP_EV   0058916864  3 Node Up
```

The number 0058916864, is the Event ID. To send this event to NetView for AIX, type the following:

```
event -E 58916864
```

The event will be displayed in the Events window.

4.7 Defining or Modifying Events

We will now examine the actions necessary to define a new event to NetView for AIX and look at ways to generate such an event.

4.7.1 Adding a New Enterprise

The ENTERPRISE field effectively identifies the application source of the trap. If you have a network management application that sends traps to NetView for AIX, then you will have to tell NetView for AIX the enterprise name, and the enterprise ID.

These values associate an enterprise ID with an enterprise-specific MIB object.

The IBM branch of the MIB tree starts with 1.3.6.1.4.1.2. Below this tree structure, you can find definitions for IBM6611, netView6000 and all other IBM MIBs. The NetView for AIX MIB Browser provides a convenient way to determine these ″dotted decimal″ values.

In the following example, two new events will be added to the NetView for AIX enterprise:

1. One to show that the SNA application on the RISC System/6000 has died
2. A second to show the application daemon has been restarted

New enterprises have to be registered with the Internet Activities Board (IAB) to ensure there are no conflicts with existing enterprises.

4.7.2 Adding New Events

The NetView for AIX enterprise name is netView6000 and the enterprise ID is 1.3.6.1.4.1.2.6.3. The generic and specific IDs are used to label individual events within an enterprise.

In this example, we have chosen to use specific value 901 for the application down event and 902 for the application restarted event.

If you require the events to be linked together using categories, NetView for AIX allows user defined categories. This option is used to link events together and makes sorting and filtering events more flexible. These are global definitions for all specific events. To define new items do the following:

- Select **Configure Categories...** from the event configuration window.
- Enter into the Category field the entry `Comms Application`.
- Select **Add** followed by **Close**.

- The new category has now been added.

  Figure 61 on page 93 shows the resulting panel.

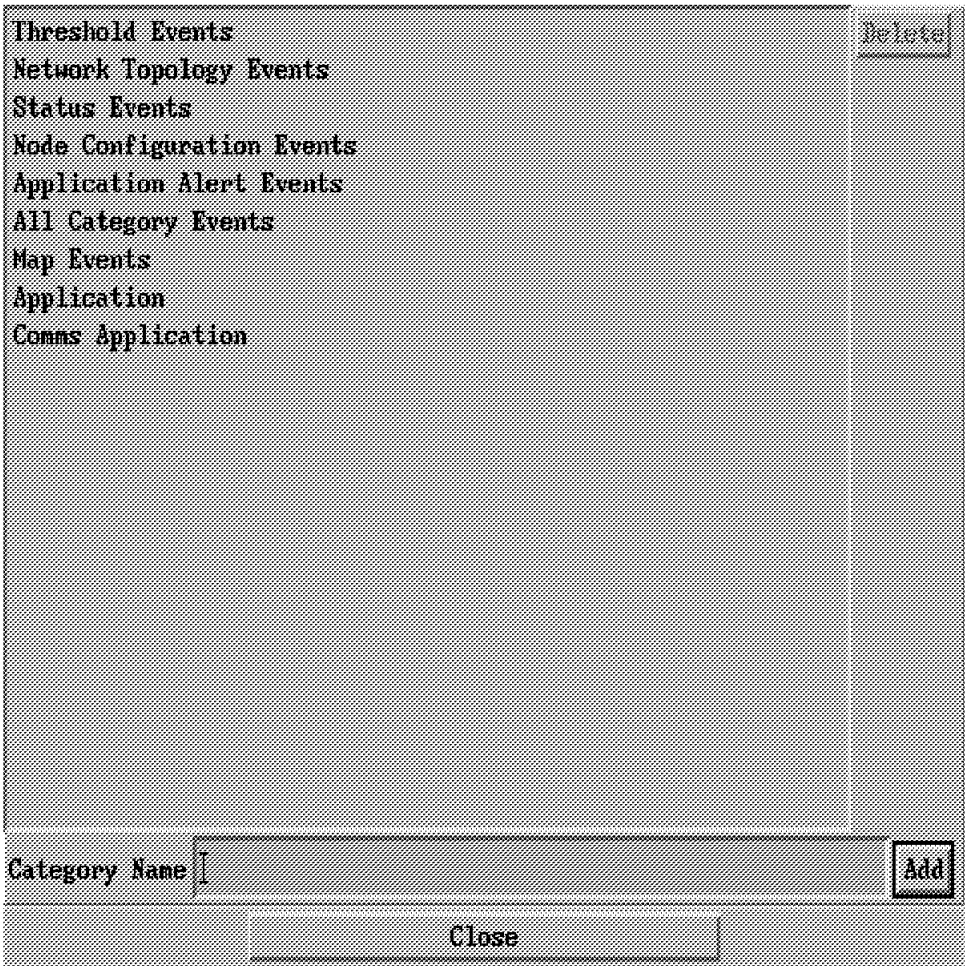


Figure 61. Configure Categories Panel

Note: The only category that will allow color status changes on the map symbols is the status events category.

The **Configure Additional Actions...** option is used in relation to the displaying events. This will be described later in this document.

To define these new events within NetView for AIX main screen, do the following:

- Select **Options->Event Configuration->Trap Customization: SNMP...** from the pull-down menus.
- Select **netView6000** enterprise located in the Event Identification window. in the **Event Identification** window. All the related events will appear in the lower half of the screen as shown in Figure 60 on page 91.
- Select **Add...** from the Event Identification window.
- Enter SNA_EV_901 in the Event Name field.
- Select **Enterprise Specific** for the generic Trap.
- Enter 901 in the Specific Trap Number field.
- Enter Application Error Detected for the Event Description.

- Enter the list (if required) for individual sources (nodes) which this event applies by keying in the hostnames followed by clicking on the Add button. In the example the nodes with SNA applications where added, these are rs60001, rs60002, rs60003 and rs60004.
- Click on Event Category and select **Status Events**.
- Click on Status and select **User 1**.
- Click on Severity and select **Critical**.
- Amend the Event Log Message to read:

  ```
  ITSO APPLICATION ERROR\nDescription - $1\nIp Address - $2\n
  Program - $3\nmachine location - $4\nContact - $5\n
  ```

  Dollar variables

  These variables: $1, $2, etc., refer to variable information that is included in the trap. This implies that we know the trap format in advance. In our case this is not a problem, since we will be creating the trap ourselves. If we were adding a trap generated by some other hardware or software we would need to refer to the manufacturer's specifications to determine the variable information enclosed in it.

  The "dollar" variables are discussed further in 4.7.3, "The Event Log Format Field" on page 96.

- Amend the Popup Notification field to read:

  ```
  901:Application Event Node $A : $1
  ```

- Enter the following text in the Command for Automatic Action field. This option will send a mail message to the root user.

  ```
  echo "SNA Error" | mailx -s "Check NetView for AIX Console" root
  ```

- Select **Ok** to update the event details.
- Select **Apply** from the Event Configuration window.

The screen will look like that in Figure 62 on page 95.

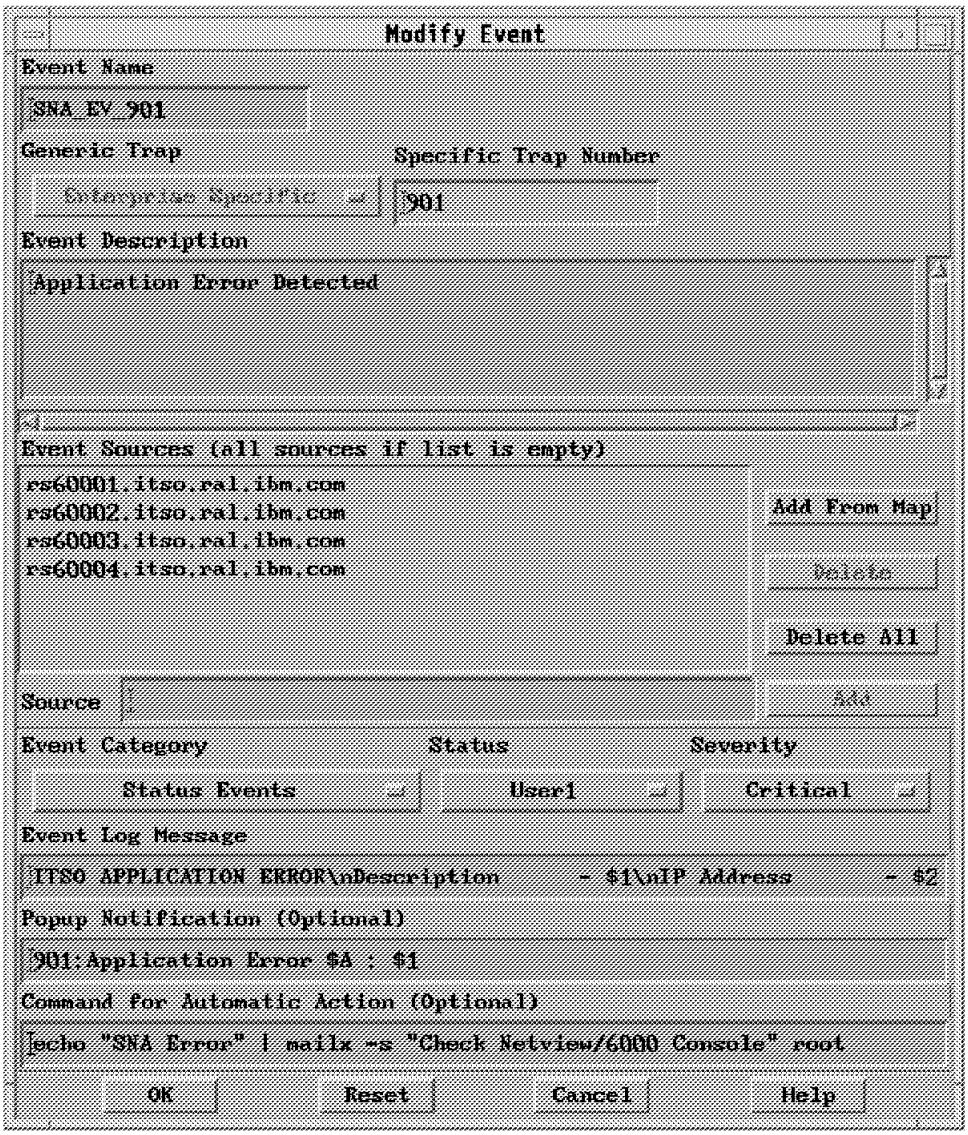


Figure 62. NetView for AIX Add/Modify Event Window

To add the second event:

1. Select **netView6000** enterprise from the Event Identification. The list of events are displayed in the **Event Identification** window.
2. Select **Add...**
3. Enter SNA_EV_902 in the Event Name field.
4. Select **Enterprise Specific**.
5. Enter 902 in the Specific Trap Number field.
6. Enter SNA APPLICATION RESTARTED for the Description.
7. Enter rs60001 for the source followed by clicking on the Add button. In the example the nodes with SNA applications where added, these are rs60001, rs60002, rs60003 and rs60004.
8. Click on Event Category and select **Status Events**.
9. Click on Status and select **Up**.

10. Click on Severity and select **Cleared**.

11. Amend the Event Log Message to read:

```
ITSO APPLICATION RESTARTED\nDescription - $1\nIp Address - $2\n
Program - $3\nmachine location - $4\nContact - $5\n
```

4.7.3 The Event Log Format Field

Any text information can be entered in this field. In this example there are five arguments passed. These will be passed from the shell script that generates the event. Other useful variables that can be used are:

```
$A        - The Network Name or IP address of the node that raised the
            event.
$G        - The Generic trap ID.
$S        - The Specific trap ID
$E        - The Enterprise as a text string
$e        - The Enterprise as an Object ID.
$1-$n     - The arguments passed to the event.
             ($3 holds useful information for events raised by the
             the netmon daemon)
*         - Displays all the variables passed to the event.
#         - Displays the number of variables passed to the event.
\n,\t     - Inserts a newline, tab respectively.

You can read further information about the use of these variables
by pressing the Help button on the Event Configuration screen.
```

Figure 63. Event Log Variables

4.7.4 The Source Field

When the source character appears on the event, it is sometimes useful to locate which program or daemon raised the event. This field is also useful when sorting or searching for specific event types. The source in the above example shows it to be from an application. Other source codes defined in NetView for AIX are:

Events Sources

```
A    - Agent
C    - xnmcollect
D    - Data collector
d    - Demo
E    - nvevents
I    - IP Map
L    - Load MIB
M    - IP topology
N    - Netmon-generated traps
n    - Netmon related
O    - OSI SA
P    - open trap (other than IP)
r    - tralertd daemon
S    - Security Agent
s    - spappld daemon
T    - trapd daemon
t    - xnmtrap
V    - Vendor
```

4.7.5 Event Customization from the Command Line

Events can also be added using the NetView for AIX addtrap command. This is useful in defining enterprise-specific traps from within a shell script or from the AIX command line.

Command example

```
/usr/OV/bin/addtrap -n netView6000          ( Enterprise name )
                    -l ITSOTEST             ( Trap Label )
                    -i 1.3.6.1.4.1.2.6.3    ( Enterprise Id )
                    -g 6                    ( Generic Id )
                    -s 903                  ( Specific Trap )
                    -o A                    ( Source-id     )
                    -t 2                    ( Type Up )
                    -c "Status Events"      ( Event Category )
                    -S 0                    ( Severity: Cleared )
                    -F "ITSO TEST $A $1 $2" ( Event Log Format )
                    -C some command         ( Optional Command  )
```

This approach is used for inserting user application-specific information into event configuration and for recovery of configured information when necessary; for example, re-adding configured traps after a system crash.

4.7.6 Sample Event Generation Shell Script

Having defined what will happen when the trap arrives, we next want to send it. The following shell script uses a function called send_trap. This script calls the AIX NetView command snmptrap to generate an event. The script will simulate the SNA daemon failing and then restarting.

```
#!/usr/bin/ksh
############################################################
# app_sendtrap using netView6000 Enterprise
# Shell to send a trap to NV/6000 from a Korn Shell script
# The trap format
#
############################################################

############################################################
#
#                Function send_app_trap
#
############################################################
function send_trap
{

TRAP_TYPE=$1       # 901 or 902 (for example)
TRAP_DEST=$2       # Destination for trap

DESCRIPTION=$3     # Application Description
SNA_NODE=$4        # Ip address of the Machine
PROG_NAME=$5       # Program Name
MACHINE_LOC=$6     # Location of the machine
CONTACT=$7         # Contact Name and Telephone Number

TRAP_AGENT=`hostname`    # Agent hostname

OCTET="octetstring" # Dummy OCTET value
MIBVAR="1.1"       # Dummy Mib Variable

/usr/OV/bin/snmptrap $TRAP_DEST .1.3.6.1.4.1.2.6.3 \
                  $TRAP_AGENT 6 $TRAP_TYPE 0 \
                  $MIBVAR $OCTET "$DESCRIPTION"  \
                  $MIBVAR $OCTET "$SNA_NODE"   \
                  $MIBVAR $OCTET "$PROG_NAME"   \
                  $MIBVAR $OCTET "$MACHINE_LOC"  \
                  $MIBVAR $OCTET "$CONTACT"

if [ "$?" -ne 0 ]
then
                  echo "snmptrap failed to send trap to $TRAP_DEST\n"
                  exit 1
fi
}
```

Figure 64 (Part 1 of 2). app_sendtrap Shell Script

```
#############################################################
#
#                 Main Body of script
#
#############################################################
clear

NETVIEWHOST=$1

# Convert $2 to standard form.

set `host $2`
SNA_HOST=$1

echo "Press <return> to Send Application Error Event \c"; read ans

send_trap "901" "$NETVIEWHOST" "Daemon Down" $SNA_HOST "SNA" \
                        "ITSC Raleigh" "919-301-1234"

echo $SNA_HOST

a /usr/OV/bin/snmptrap $TRAP_DEST .1.3.6.1.4.1.2.6.3.1 \
  $TRAP_AGENT 6 58916871 1 \
  .1.3.6.1.4.1.2.6.3.1.1.2.0 Integer 14 \
  .1.3.6.1.4.1.2.6.3.1.1.3.0 OctetString "$SNA_HOST" \
  .1.3.6.1.4.1.2.6.3.1.1.4.0 OctetString "Object status is" \
  .1.3.6.1.4.1.2.6.3.1.1.5.0 OctetString Down


echo "Press <return> to Send Application Restored Event \c"; read ans

send_trap "902" "$NETVIEWHOST" "Daemon Restored" $SNA_HOST "SNA" \
                        "ITSC Raleigh" "919-301-1234"

a /usr/OV/bin/snmptrap $TRAP_DEST .1.3.6.1.4.1.2.6.3.1 \
  $TRAP_AGENT 6 58916871 1 \
  .1.3.6.1.4.1.2.6.3.1.1.2.0 Integer 14 \
  .1.3.6.1.4.1.2.6.3.1.1.3.0 OctetString "$SNA_HOST" \
  .1.3.6.1.4.1.2.6.3.1.1.4.0 OctetString "Object status is" \
  .1.3.6.1.4.1.2.6.3.1.1.5.0 OctetString Up


echo "2 Events sent to $NETVIEWHOST......."
```

Figure 64 (Part 2 of 2). app_sendtrap Shell Script

To execute the script, type:

```
app_sendtrap <NetView for AIX host name> <Node with SNA>
```

The following was used for this example:

```
app_sendtrap rs60003 rs600010
```

Press Enter once. Two events are displayed as seen in Figure 66 on page 101. You will see a pop-up window appear and the symbol representing the relevant machine change color, if:

- The submap which contains the relevant machine has the symbol representing the machine (NetView for AIX object) controlled by Status Source: Symbol.

  The setting of the machine color is done via the snmptrap shown in **a**.

The pop-up is displayed in Figure 69 on page 103.

Status Source

The color of symbols on NetView for AIX displays is controlled by the Status Source. You can see the Status Source by selecting a symbol with the right mouse button and following **Edit->Modify/Describe->Symbol**. For the above snmptrap to affect the color, the Status Source must be: Symbol.

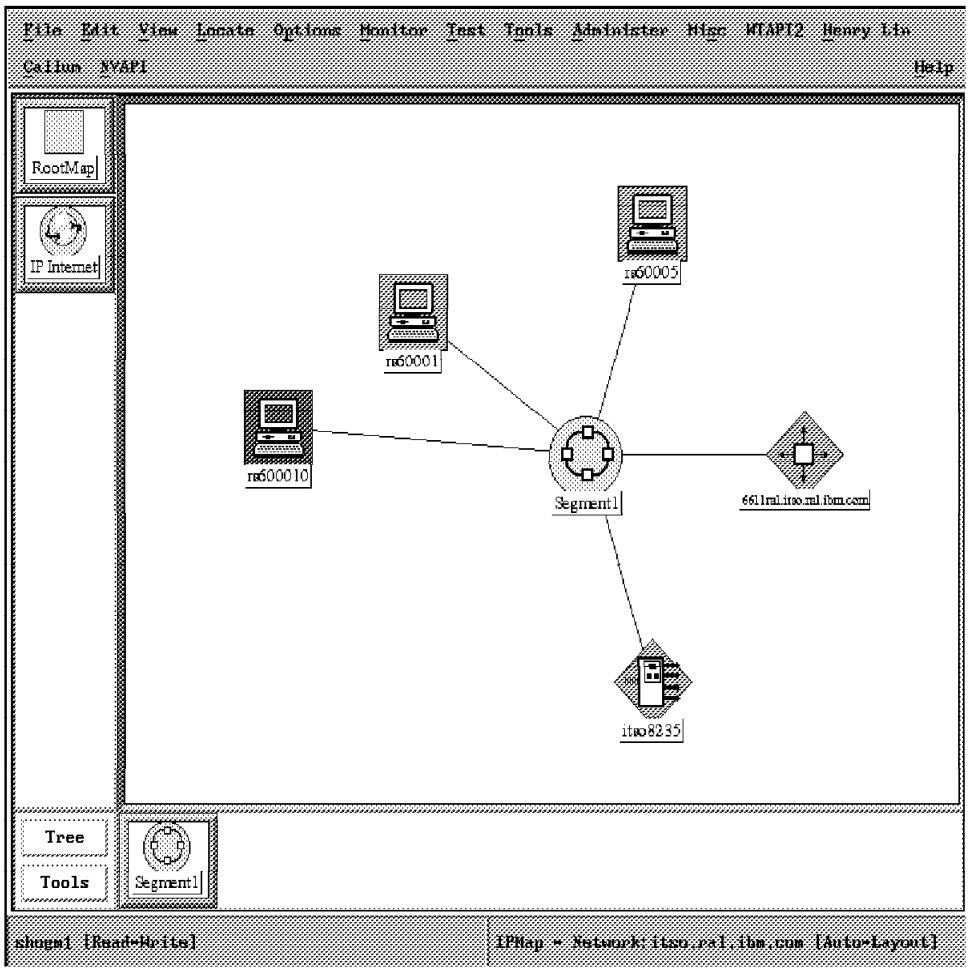


Figure 65. rs600010 After snmptrap Set Object Down. rs600010 was controlled in this submap by Status Source: Symbol. This allowed the snmptrap to change the symbol's color.

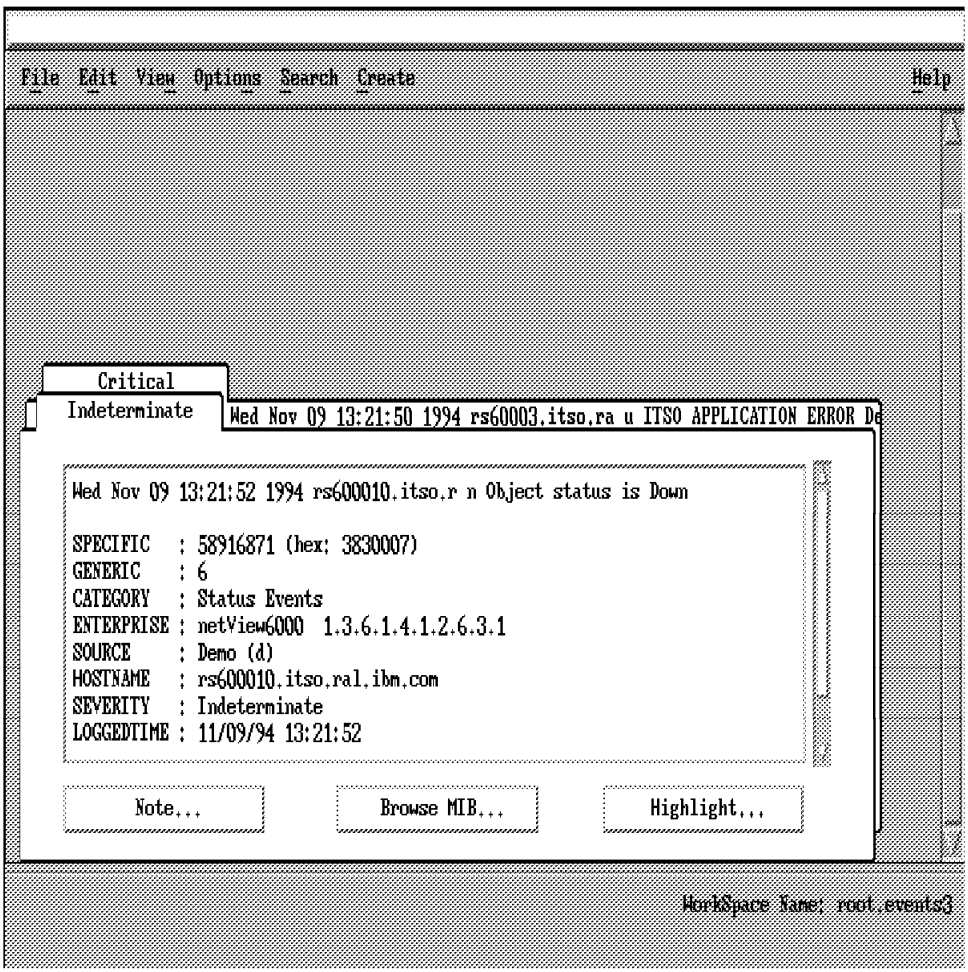


Figure 66. NetView for AIX Selected Events - Part 1

Press Enter once more, and two more events will be displayed in the Events window as shown in Figure 68 on page 103. This time no warning screen will appear. The status of the symbol will now turn back to GREEN for the same reasons as discussed previously.

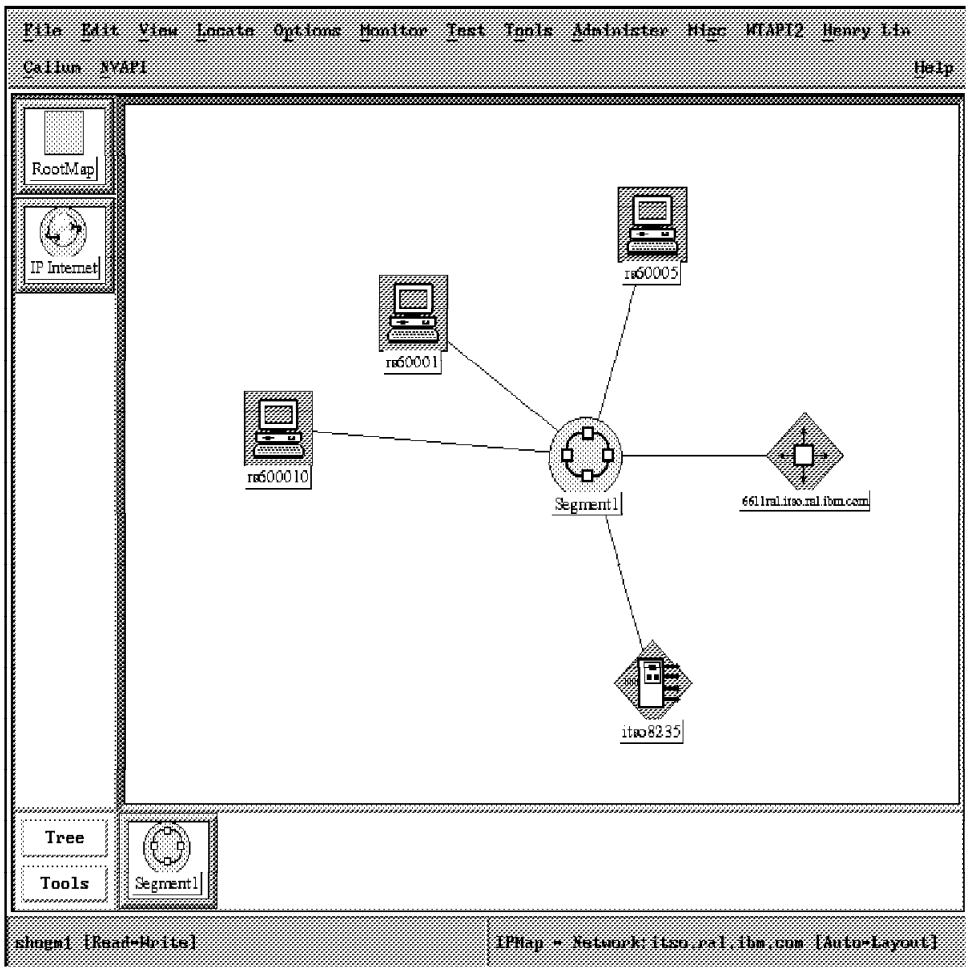


Figure 67. rs600010 After snmptrap Set Object Up. rs600010 was controlled in this submap by Status Source: Symbol. This allowed the snmptrap to affect the symbol's color.

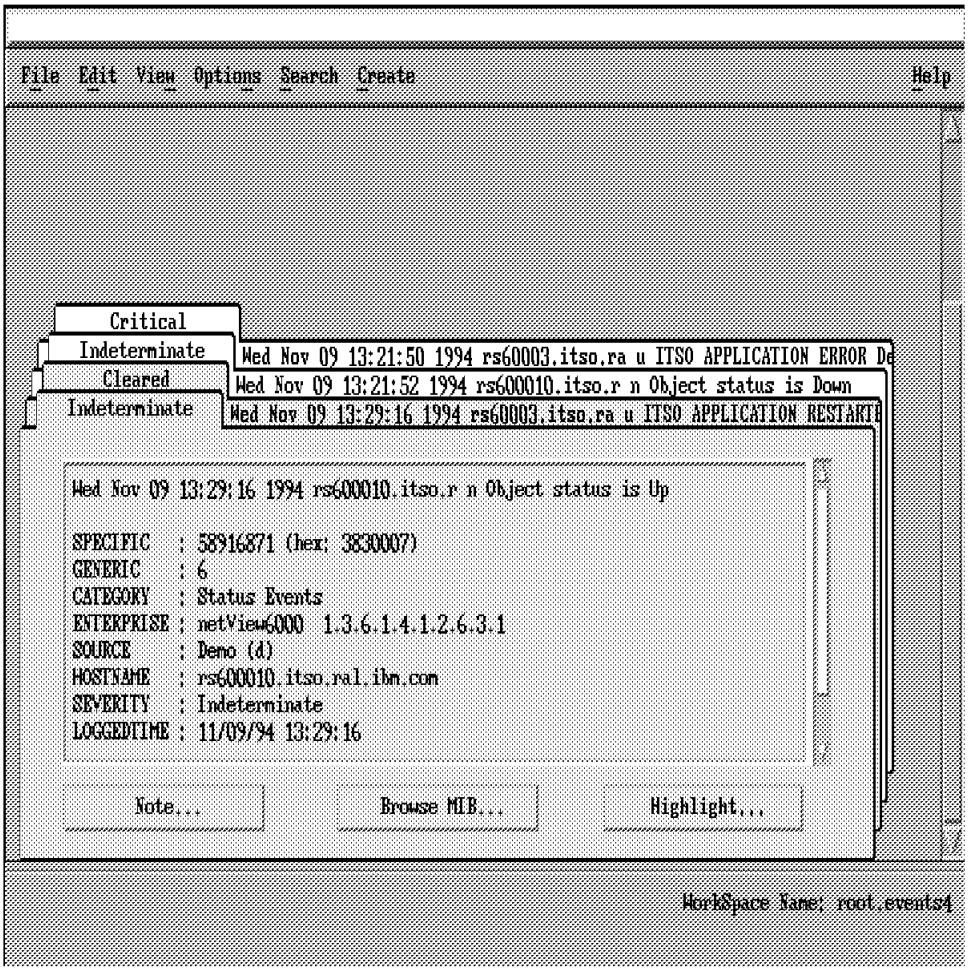


Figure 68. NetView for AIX Selected Events - Part 2

Figure 69. NetView for AIX Warning Screen

4.7.7 Status Source and User Symbols

This section further discusses the status source fields mentioned in 4.7.6, “Sample Event Generation Shell Script” on page 97 and also discusses an approach for adding user symbols to a submap via a sample set of code which utilizes the NetView for AIX EUI API.

4.7.7.1 Symbol Status Source

In section 4.7.6, “Sample Event Generation Shell Script” on page 97 a symbol’s Status Source was mentioned. If this field, for particular submaps which have Status Source: Symbol, the snmptrap:

```
/usr/OV/bin/snmptrap rs60003 .1.3.6.1.4.1.2.6.3.1 rs60003 6 58916871 1\
  .1.3.6.1.4.1.2.6.3.1.1.2.0 Integer 14 \
  .1.3.6.1.4.1.2.6.3.1.1.3.0 OctetString "rs600010.itso.ral.ibm.com" \
  .1.3.6.1.4.1.2.6.3.1.1.4.0 OctetString "Object status is" \
  .1.3.6.1.4.1.2.6.3.1.1.5.0 OctetString Down
```

will affect the color of the displayed symbol for the object: rs600010.itso.ral.ibm.com. The Status Source must be Symbol for this to take affect. Using this trap will affect NetView for AIX-controlled (IP) resources. NetView for AIX, through its normal processes may set the symbol back to its discovered status or the user may issued a trap such as:

```
/usr/OV/bin/snmptrap rs60003 .1.3.6.1.4.1.2.6.3.1 rs60003 6 58916871 1\
  .1.3.6.1.4.1.2.6.3.1.1.2.0 Integer 14 \
  .1.3.6.1.4.1.2.6.3.1.1.3.0 OctetString "rs600010.itso.ral.ibm.com" \
  .1.3.6.1.4.1.2.6.3.1.1.4.0 OctetString "Object status is" \
  .1.3.6.1.4.1.2.6.3.1.1.5.0 OctetString Up
```

to further modify the symbol’s color.

The snmptrap above can be used to control IP status; but, remember, when so doing you are affecting NetView for AIX-managed resources. One possible reason for doing this may be to include user (or non-IP) status in graphical presentations which an operator may be reviewing as a normal process. This combination of IP and non-IP is addressed by the gtm and open topology support of NetView for AIX which is discussed later on in this document. However, for the example in this section we will use the graphical EUI API support in NetView for AIX.

First, let us look at two separate submaps and see the Status Source:

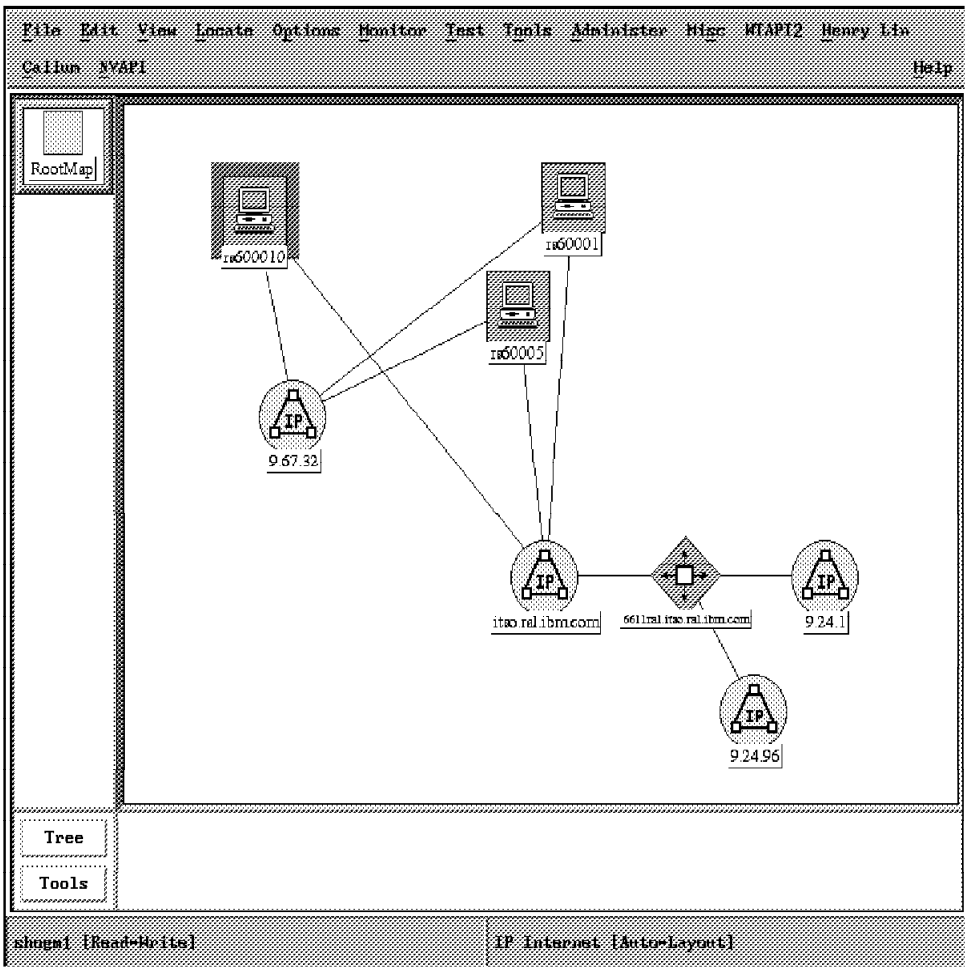


Figure 70. IP Internet Submap

In the above submap, right-button clicking on rs600010 leads to the following:

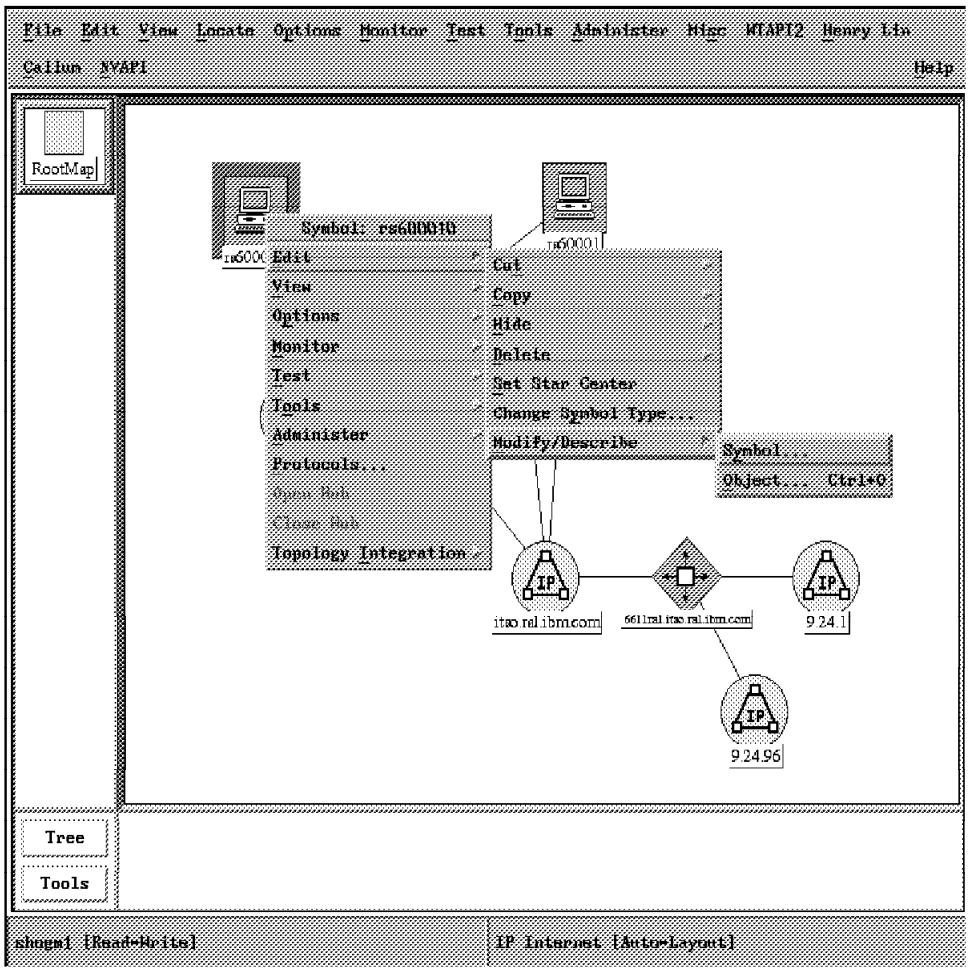


Figure 71. IP Internet Submap Heading Toward Symbol

Clicking on Symbol presents the following panel:

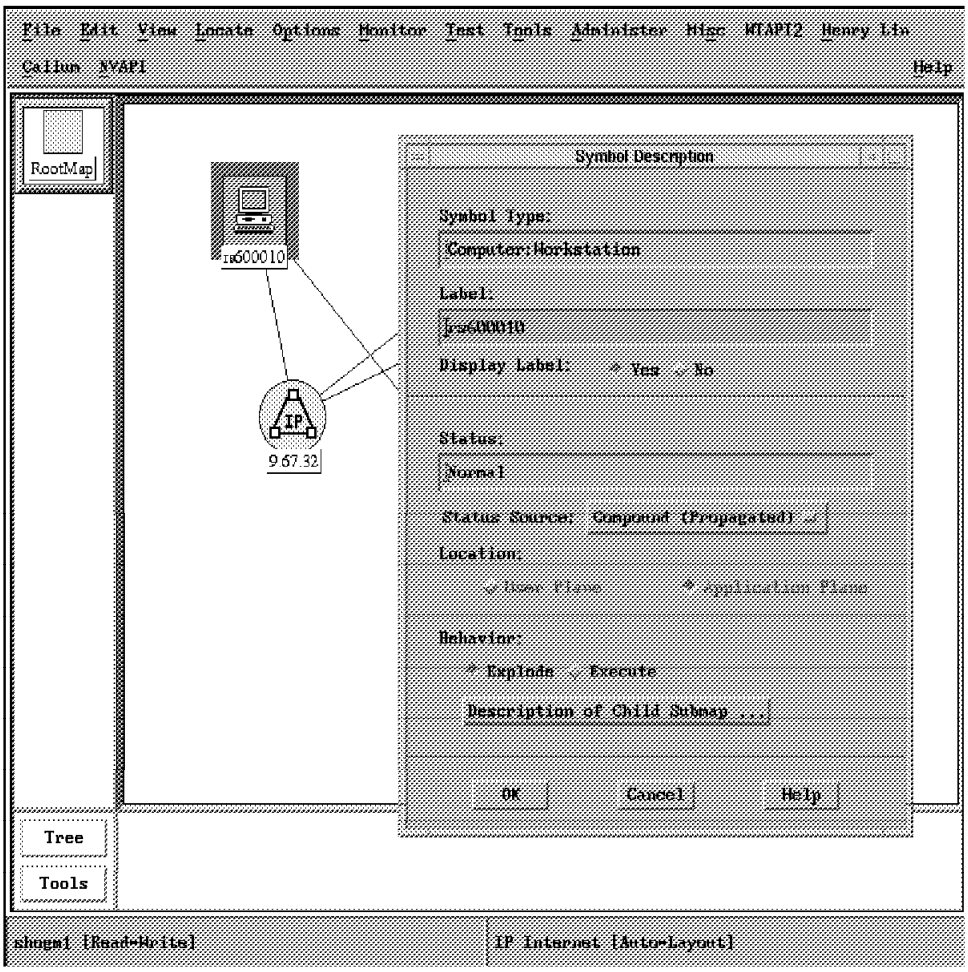


Figure 72. IP Internet Submap Symbol Description

Status Source: Compound (Propagated) would not result in this submap's symbol being affected by the previously discussed snmptrap.

A second submap is in the following figure.

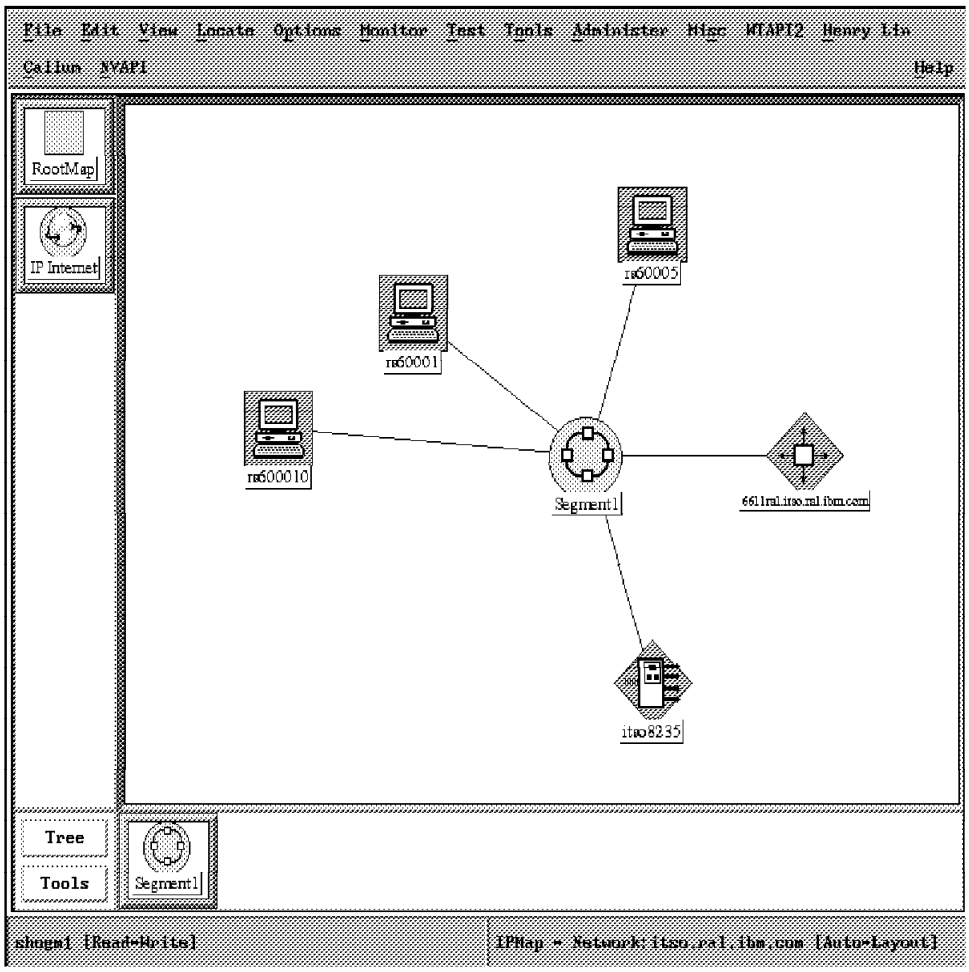


Figure 73. IPMap - Network: Submap

In the above submap, right-button clicking on rs600010 and clicking on Symbol presents the following panel:

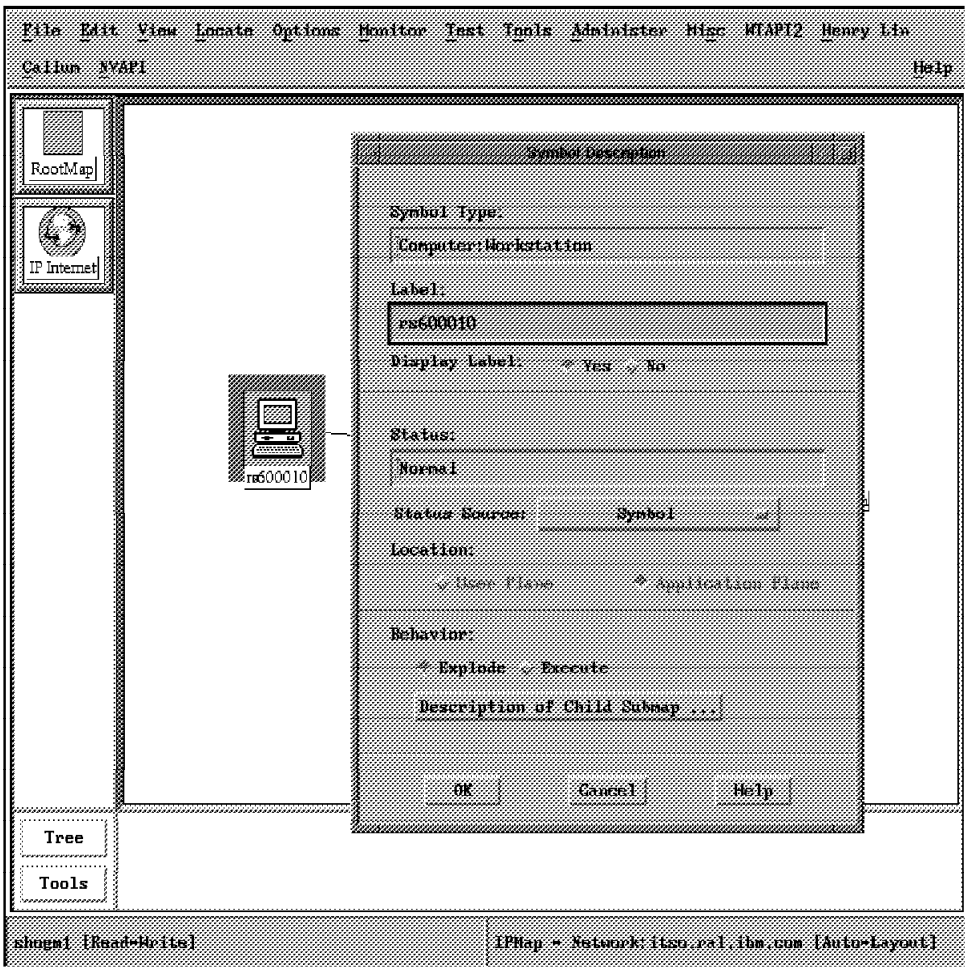


Figure 74. IPMap - Network: Submap Symbol Description

Status Source: Symbol would result in this submap's symbol being affected by the previously discussed snmptrap.

The trap could be triggered in a shell such as in the following figure.

Note: You are affecting IP status of a NetView for AIX-managed object. This may result in the operator having a view of an object's status which is not as expected from a hardware or similar point of view until the status is restored by NetView for AIX discovery or the user.

Examples of using the following shell are:

```
set_ip_status  rs60003  rs600010  Down

set_ip_status  rs60003  rs600010  Up
```

```
#!/usr/bin/ksh
#
# set_ip_status
#  Used to set IP status.  It will affect color on symbols controlled
#                          by Status Source: Symbol
#
#  Note: You are affecting IP-managed objects!
#
#  Parameters: $1 = Manager Machine
#              $2 = Symbol name (the following `host`
#                                resolves the object name
#              $3 = Status   such as Up, Down, User1, etc.
#
#

clear

NETVIEWHOST=$1
FUNCTION=$3

# Convert $2 to standard form.

set `host $2`
SNA_HOST=$1


/usr/OV/bin/snmptrap $NETVIEWHOST .1.3.6.1.4.1.2.6.3.1 \
   $NETVIEWHOST 6 58916871 1 \
   .1.3.6.1.4.1.2.6.3.1.1.2.0 Integer 14 \
   .1.3.6.1.4.1.2.6.3.1.1.3.0 OctetString "$SNA_HOST" \
   .1.3.6.1.4.1.2.6.3.1.1.4.0 OctetString "Object status is" \
   .1.3.6.1.4.1.2.6.3.1.1.5.0 OctetString $FUNCTION
```

Figure 75. set_ip_status

It is possible that, for the example discussed in Section 4.7.6, “Sample Event Generation Shell Script” on page 97 the user may choose to include a non-IP symbol in an IP-managed Submap. This could be done in a number of ways including the normal NetView for AIX operator pull-downs such as Edit/Add/Object.

In the following example the user-written implementation of NetView for AIX graphics EUI API support: wtdriver6/wteuiap6 is shown. Refer to Chapter 7, “wtdriver6/wteuiap6 Sample NetView for AIX EUI API” on page 217 for an overview of this user-written code.

In the example discussed in Section 4.7.6, “Sample Event Generation Shell Script” on page 97 it was indicated that an application (SNA) was simulated up/down.

The operator may benefit from seeing on the submap for the IP node used in this example (rs600010) a symbol representing "sna". The following figures show this.

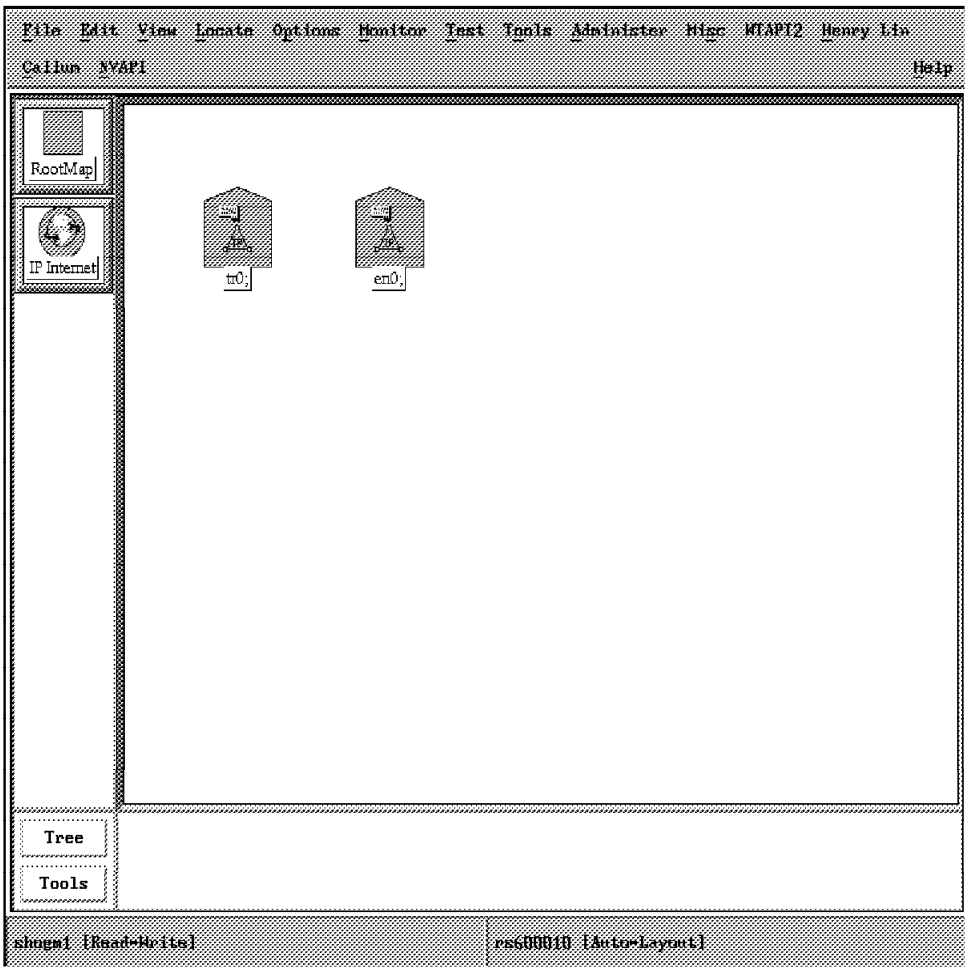


Figure 76. rs600010 Submap Without sna Symbol

Issuing the following commands add the "sna" symbol to the submap and sets the added symbol "up" to give it a normal color. wtdriver6 calls wteuiap6 and executes the following requests.

```
wtdriver6 -f rs600010 add sna ap
wtdriver6 -f rs600010 set sna up
```

The "-f" flag tells wtdriver6/wteuiap6 to focus on the rs600010 submap for the particular command. The following figure shows the resultant submap.

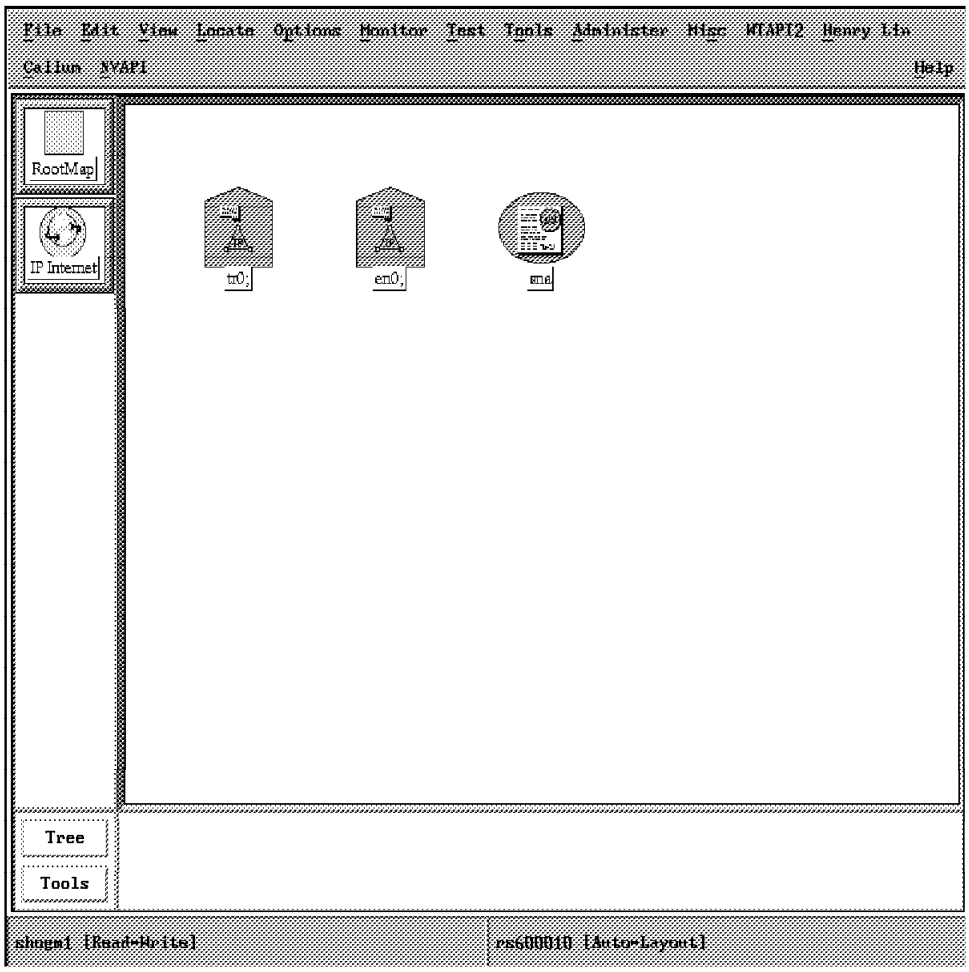


Figure 77. rs600010 Submap with sna Symbol

If the previously discussed shell was modified to include:

```
wtdriver6 -f rs600010 set sna up
```

or

```
wtdriver6 -f rs600010 set sna down
```

The sna symbol would reflect the particular color/status and rs600010 submaps which had Status Source: Compound (Propagated) (see Figure 72 on page 107) would be affected.

In addition, it is possible that the user would like to enter a user text field associated with the "sna" object in the NetView for AIX object database. The following command is an example of doing this:

```
wtdriver6 assoc rs600010 "Some String" "sna is managed by John Anderson,
 Call: 301-2308"
```

Issuing the normal NetView for AIX command ovobjprint results in the following:

```
ovobjprint -s sna

FIELD ID        FIELD NAME          FIELD VALUE
10              Selection Name      "sna"
14              OVW Maps Exists     1
15              OVW Maps Managed    1
66              isSoftware          TRUE
129             Software Status     "down"
135             Some String         "sna is managed by John Anderson,
                                     Call: 301-2308"
```

The example discussed in this section can be used to show one level of integration between IP and non-IP resources. Another approach is discussed later in this document using gtm and open topology support in NetView for AIX.

4.7.8 Adding a New Enterprise with an Associated Event

In the previous example, the enterprise for netView6000 was used. The enterprise was: 1.3.6.1.4.1.2.6.3.1 and snmptrap issued traps within that enterprise. Generic type 6, specific trap numbers 901 and 902 were defined by the user and trap number 58916871 was pre-defined by NetView for AIX.

Instead of using the netView6000 enterprise, the following example shows how to define a new enterprise and add a specific event. The ITSO in Raleigh has an authorized MIB definition enterprise Id of **1.3.6.1.4.1.2.8.1**. The example event defined below is used for reporting DCE errors generated on the DCE server. To add the new configuration items do the following:

- Select **Options->Event Configuration->Trap Customization...** from the NetView for AIX pull-down menu.
- Select **Add...** from the Enterprise Identification window.
- Enter `itso_raleigh` in the Enterprise Name field.
- Enter `1.3.6.1.4.1.2.8.1` for the enterprise Id.
- Select **Add**.
- Click on **itso_raleigh** enterprise.
- Select **Add...** from the Event Identification window.
- Select **Add...** from the Event Identification window.
- Enter `DCE_910` in the Event Name field.
- Select **Enterprise Specific** for the generic Trap.
- Enter 910 in the Specific Trap Number field.
- Enter `DCE Agent Error` for the Event Description.
- Leave the source field blank.
- Click on Event Category and select **Application**.
- Click on Status and select **Default Status**.
- Click on Severity and select **Major**.
- Amend the Event Log Message to read:

  `ITSO DCE ERROR\nDCE Cell Name - $1\nDCE Agent - $2\n`
- Select **Ok** and apply from the Event Configuration window.

The event is now configured. To test the event type in the following command:

```
snmptrap `hostname` .1.3.6.1.4.1.2.8.1 `hostname` 6 910 0 \
1.1 octetstring `getcellname`    \
1.1 octetstring "Security"
```

The event will appear as shown in Figure 78.

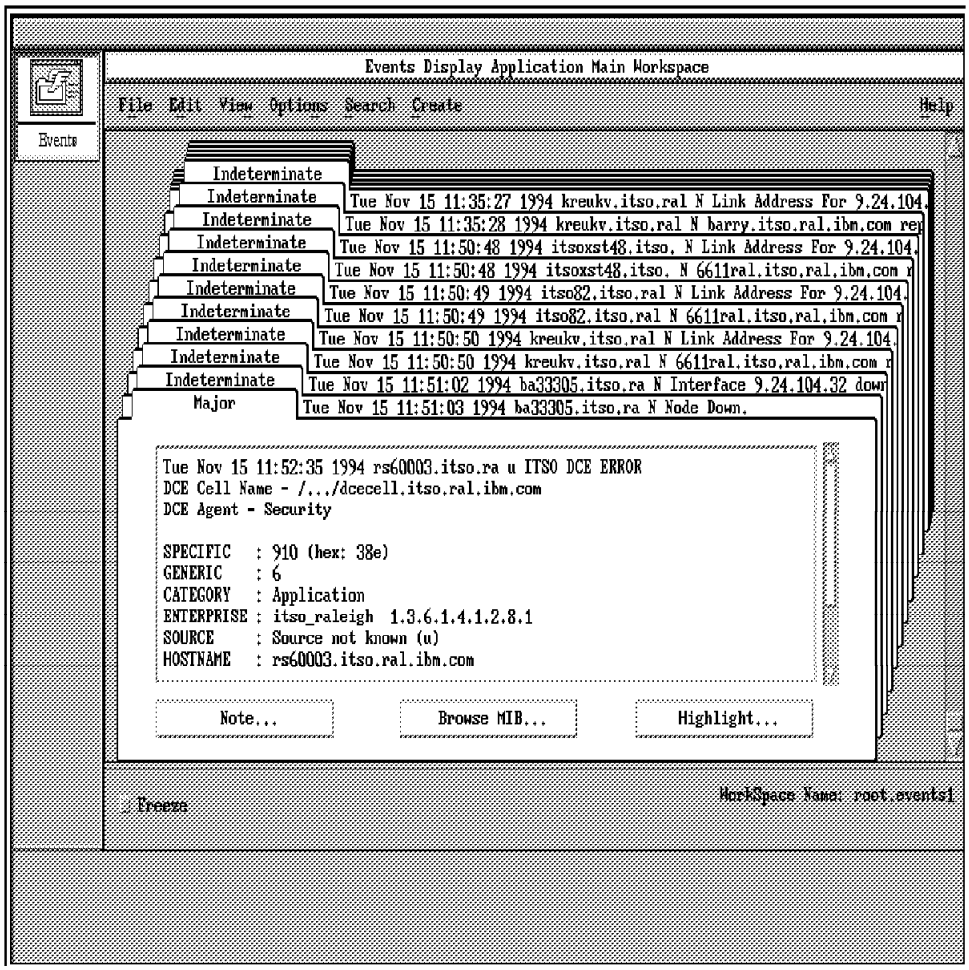


Figure 78. AIX DCE Event Display

4.7.8.1 Example of Using the ITSO Enterprise

In section 4.7.6, "Sample Event Generation Shell Script" on page 97 an example was shown using snmptrap to interface with NetView for AIX via events used within its enterprise.

The following example is the same application but using the ITSO enterprise. The shell script uses a function called send_trap and, as shown in the following figure, the enterprise used in the snmptrap to generate an event was modified for the ITSO enterprise **a**. Other behavior of the previous example is the same. Note that the shell continues to use the netView6000 enterprise **b** for the trap which controls the color of the NetView for AIX-managed object.

```
#!/usr/bin/ksh
############################################################
# app_sendtrap using ITSO Enterprise
# Shell to send a trap to NV/6000 from a Korn Shell script
# The trap format
#
############################################################

############################################################
#
#                Function send_app_trap
#
############################################################
function send_trap
{

TRAP_TYPE=$1       # 901 or 902 (for example)
TRAP_DEST=$2       # Destination for trap

DESCRIPTION=$3     # Application Description
SNA_NODE=$4        # Ip address of the Machine
PROG_NAME=$5       # Program Name
MACHINE_LOC=$6     # Location of the machine
CONTACT=$7         # Contact Name and Telephone Number

TRAP_AGENT=`hostname`    # Agent hostname

OCTET="octetstring" # Dummy OCTET value
MIBVAR="1.1"       # Dummy Mib Variable
#                               Use ITSO Enterprise
#
 a  /usr/OV/bin/snmptrap $TRAP_DEST .1.3.6.1.4.1.2.8.1 \
                  $TRAP_AGENT 6 $TRAP_TYPE 0 \
                 $MIBVAR $OCTET "$DESCRIPTION"  \
                 $MIBVAR $OCTET "$SNA_NODE"   \
                 $MIBVAR $OCTET "$PROG_NAME"   \
                 $MIBVAR $OCTET "$MACHINE_LOC"  \
                 $MIBVAR $OCTET "$CONTACT"

if [ "$?" -ne 0 ]
then
                 echo "snmptrap failed to send trap to $TRAP_DEST\n"
                 exit 1
fi
}
```

Figure 79 (Part 1 of 2). app_sendtrap_itso_enterprise Shell Script

```
############################################################
#
#                 Main Body of script
#
############################################################
clear

NETVIEWHOST=$1

# Convert $2 to standard form.

set `host $2`
SNA_HOST=$1

echo "Press <return> to Send Application Error Event \c"; read ans

send_trap "901" "$NETVIEWHOST" "Daemon Down" $SNA_HOST "SNA" \
                        "ITSC Raleigh" "919-301-1234"

echo $SNA_HOST

 b  /usr/OV/bin/snmptrap $TRAP_DEST .1.3.6.1.4.1.2.6.3.1 \
   $TRAP_AGENT 6 58916871 1 \
   .1.3.6.1.4.1.2.6.3.1.1.2.0 Integer 14 \
   .1.3.6.1.4.1.2.6.3.1.1.3.0 OctetString "$SNA_HOST" \
   .1.3.6.1.4.1.2.6.3.1.1.4.0 OctetString "Object status is" \
   .1.3.6.1.4.1.2.6.3.1.1.5.0 OctetString Down


echo "Press <return> to Send Application Restored Event \c"; read ans

send_trap "902" "$NETVIEWHOST" "Daemon Restored" $SNA_HOST "SNA" \
                        "ITSC Raleigh" "919-301-1234"

 b  /usr/OV/bin/snmptrap $TRAP_DEST .1.3.6.1.4.1.2.6.3.1 \
   $TRAP_AGENT 6 58916871 1 \
   .1.3.6.1.4.1.2.6.3.1.1.2.0 Integer 14 \
   .1.3.6.1.4.1.2.6.3.1.1.3.0 OctetString "$SNA_HOST" \
   .1.3.6.1.4.1.2.6.3.1.1.4.0 OctetString "Object status is" \
   .1.3.6.1.4.1.2.6.3.1.1.5.0 OctetString Up


echo "2 Events sent to $NETVIEWHOST......."
```

Figure 79 (Part 2 of 2). app_sendtrap_itso_enterprise Shell Script

The script was executed the same way as previously discussed:

```
app_sendtrap <NetView for AIX host name> <Node with SNA>
```

The following was used for this example:

```
app_sendtrap rs60003 rs600010
```

The following figures show information related to this example.

First, the ITSO enterprise and specific traps have been added to NetView for AIX using event configuration support.

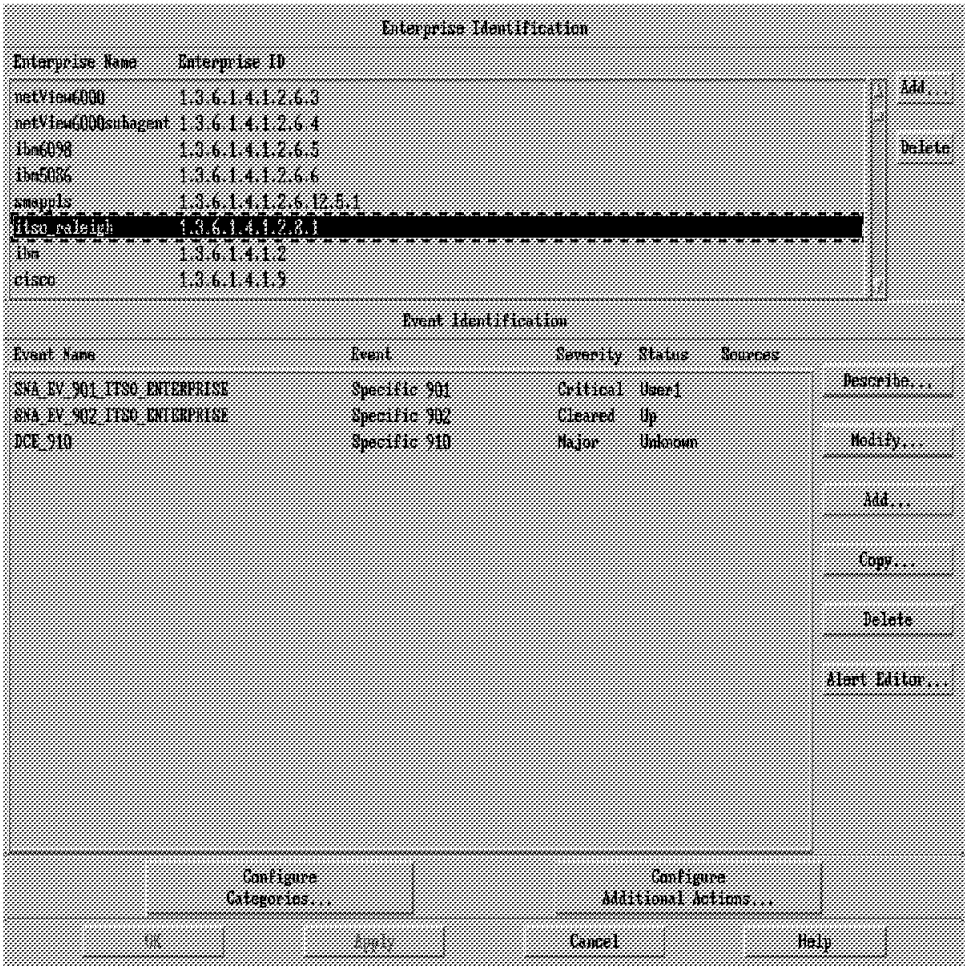


Figure 80. itso.raleigh Enterprise and Specific Events Used in this Example

If the enterprise and/or specific traps arrive at NetView for AIX without being defined, information to the operator would show up as in the following figure.

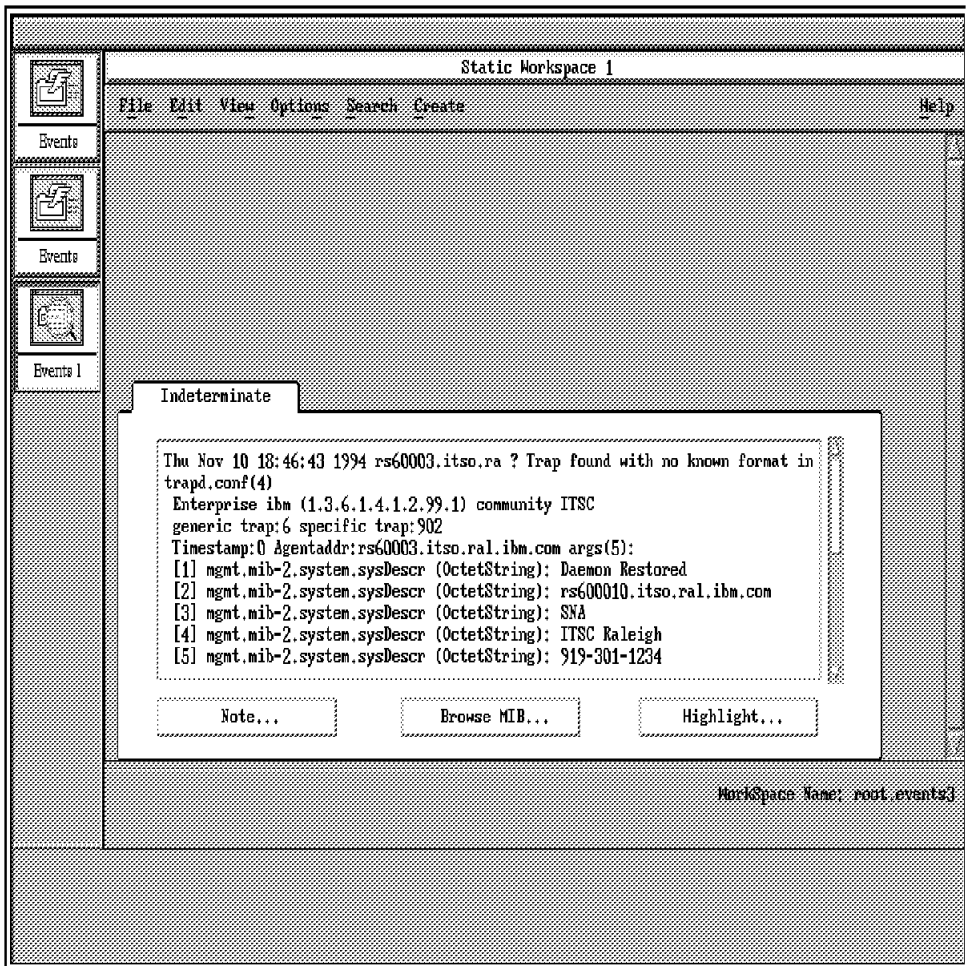


Figure 81. Unknown Trap Arrived at NetView for AIX

The following figure shows the two itso.raleigh enterprise traps from this example.

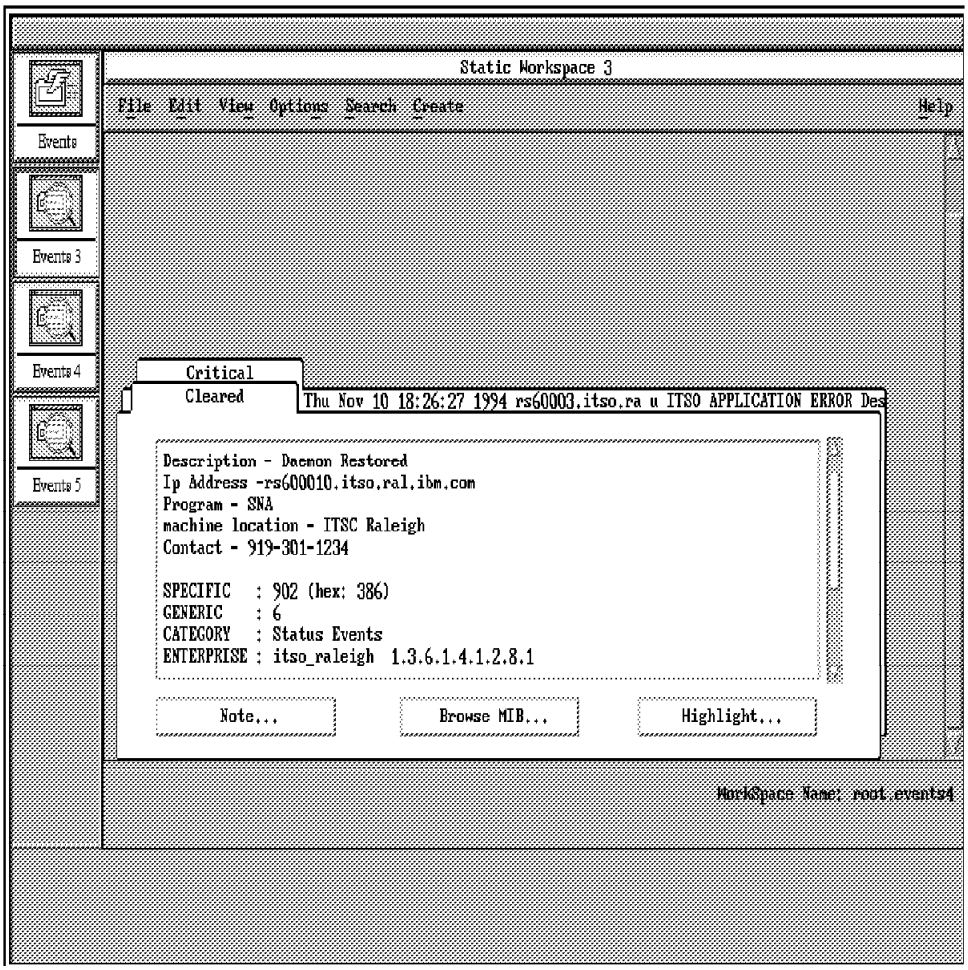


Figure 82. itso.raleigh Enterprise and Specific Events Arrive

The following figure shows the two previously-discussed netView6000 traps which were used to set the NetView for AIX-managed resource Down and Up.

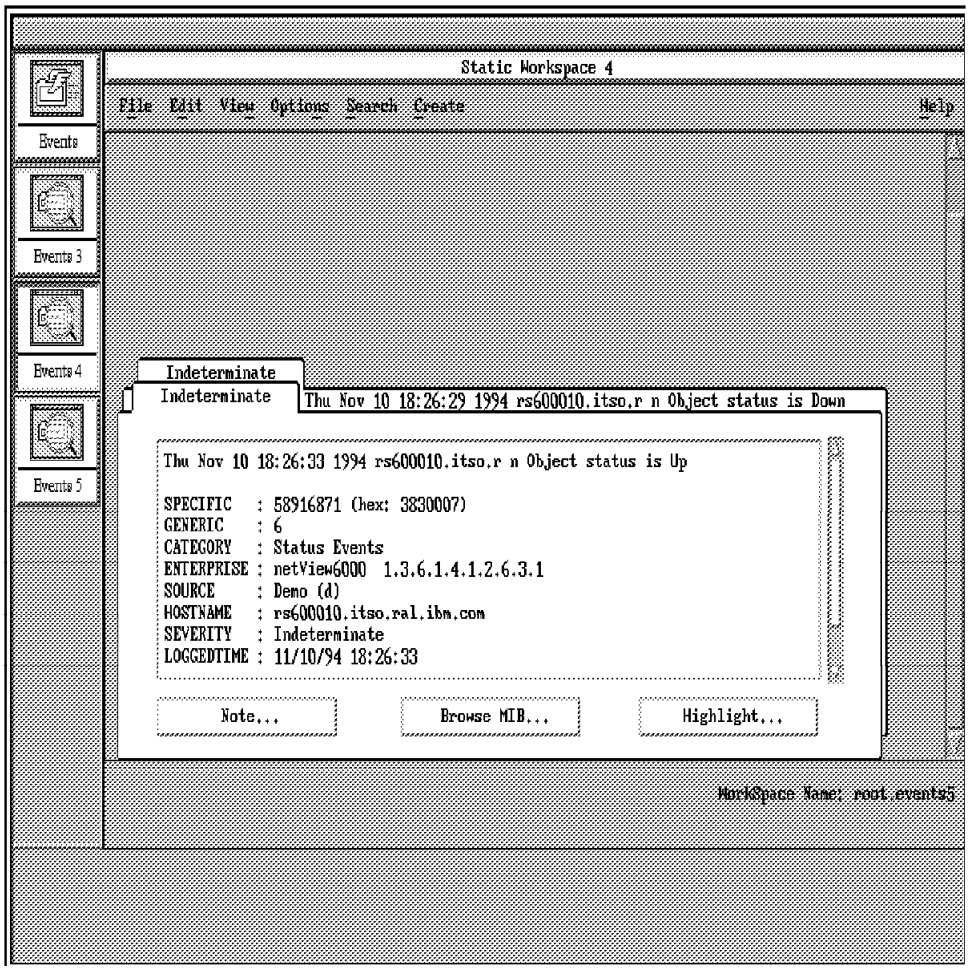


Figure 83. netView6000 Enterprise and Specific Events Arrive

The following figure shows the four traps involved in this example.

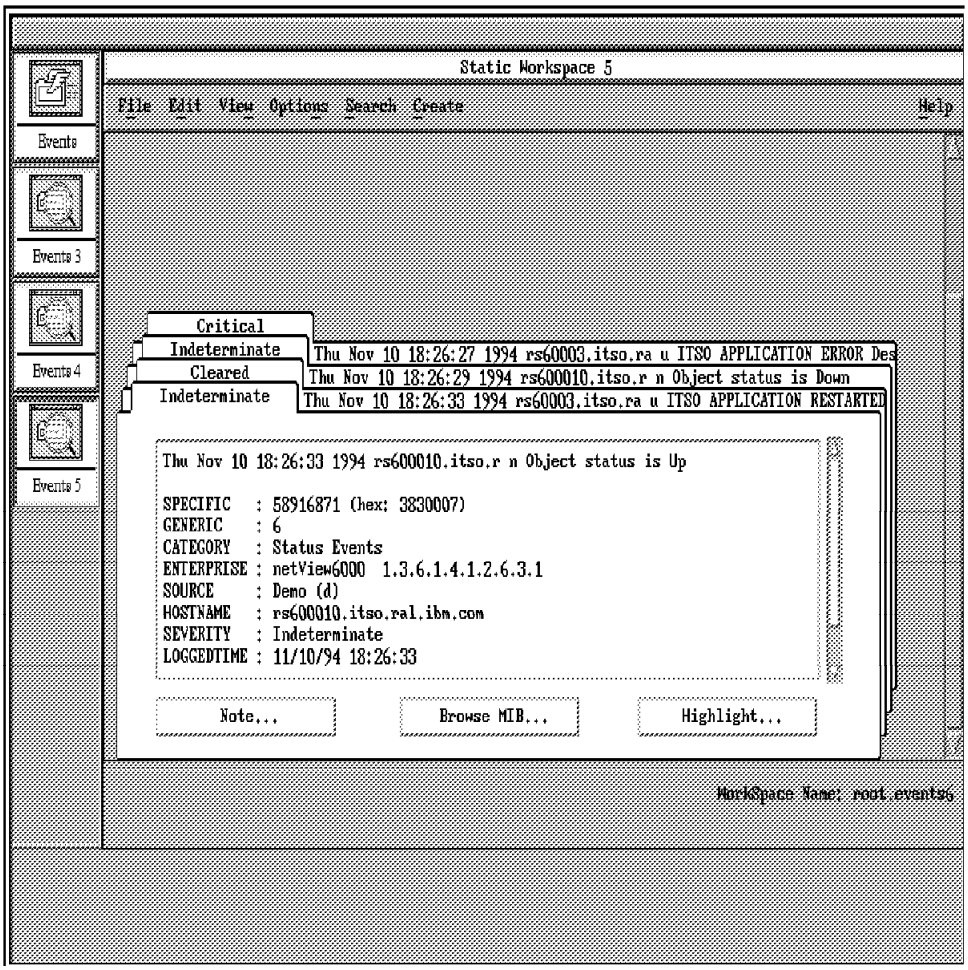


Figure 84. netView6000 Enterprise and itso.raleigh Specific Events Arrive

In some cases (for example if an installed product is part of the NetView Association) an application's enterprise and specific traps plus, possibly, its MIB description will be included as normal support in NetView for AIX. If not, it is the user's responsibility to use event configuration and MIB loader support to accomplish these acts.

4.8 NetView for AIX Filters

If a large number of events are being received by NetView for AIX, it may be necessary to filter out the unwanted ones.

Event filters are configured for a number of reasons, such as:

- To select which events are displayed by the nvevents application
- To select which events are displayed in each dynamic workspace
- To select which events are converted into alerts and sent to S/390 NetView

NetView for AIX has two types of filters, simple and compound. The simple filter is an expression that includes SNMP criteria, whereas the compound filter is composed of a number of simple filter expressions, and uses the logical operators AND, OR and NOT. Both filter types are edited using the NetView for AIX filter editor executed from the pull-down menu. You should not amend the

filter file directly using an AIX editor such as vi. When multiple simple filters are activated they use the logical OR operator. If you are using more than one filter at a time, you should create a combined filter for more effective trap exclusion.

The filter configuration files are located in the directory /usr/OV/filters. The default file is called filter.samples.

The NetView for AIX filtering option supplies the following functions:

- Filtering by event content criteria
- Filtering by event frequency
- Filtering by hostname or IP address
- Control of filter activation by start and stop times

4.8.1 Filter Editor Screen

To start the filter editor application, do one of the following:

- Select **Tools -> Filter Editor**, from the AIX NetView for AIX pull-down menu

  or

- Type /usr/OV/bin/filtered from the command line.

The screen will look similar to Figure 85.

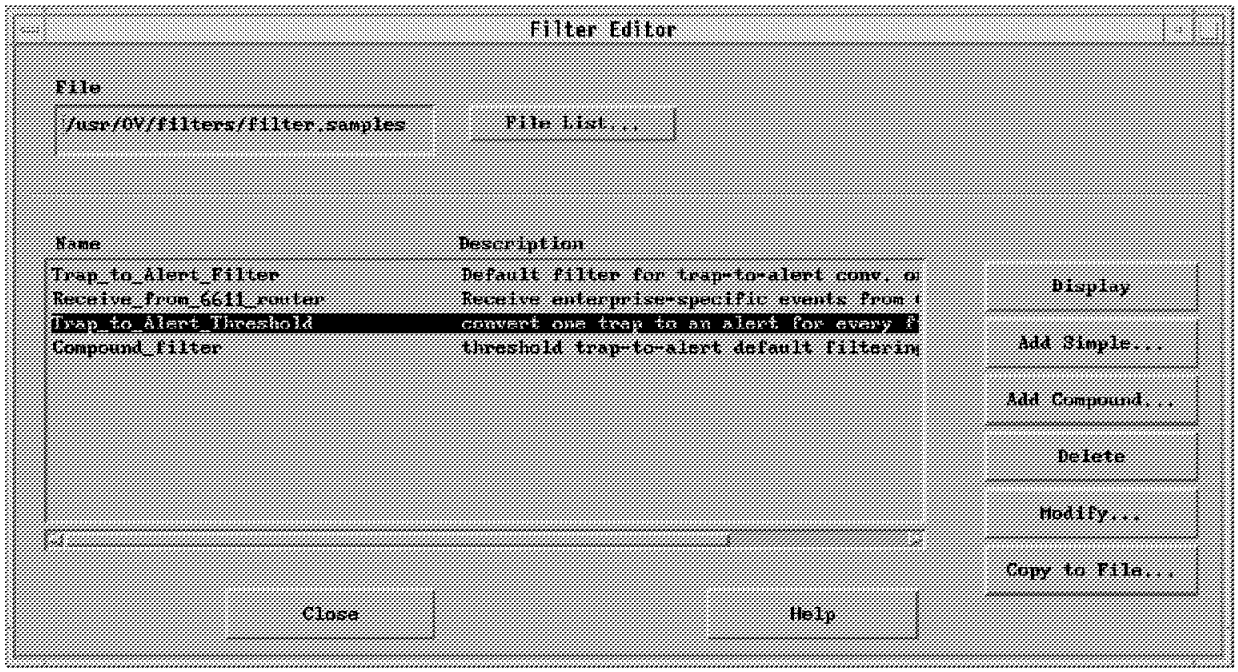


Figure 85. Filter Editor Selection Screen

For this example, we will create a new filter to stop the event "902" from being displayed on the Events window. Follow the procedure below:

- Select **Add Simple**.
- Enter the Filter Name: application_started.
- Enter the Description: Filter for Application Started Events.
- Select **Events not Equal to Selected**.

- Select **Add/Modify...**

  The Enterprise Specific Trap Selection menu now appears, as can be seen in Figure 86.

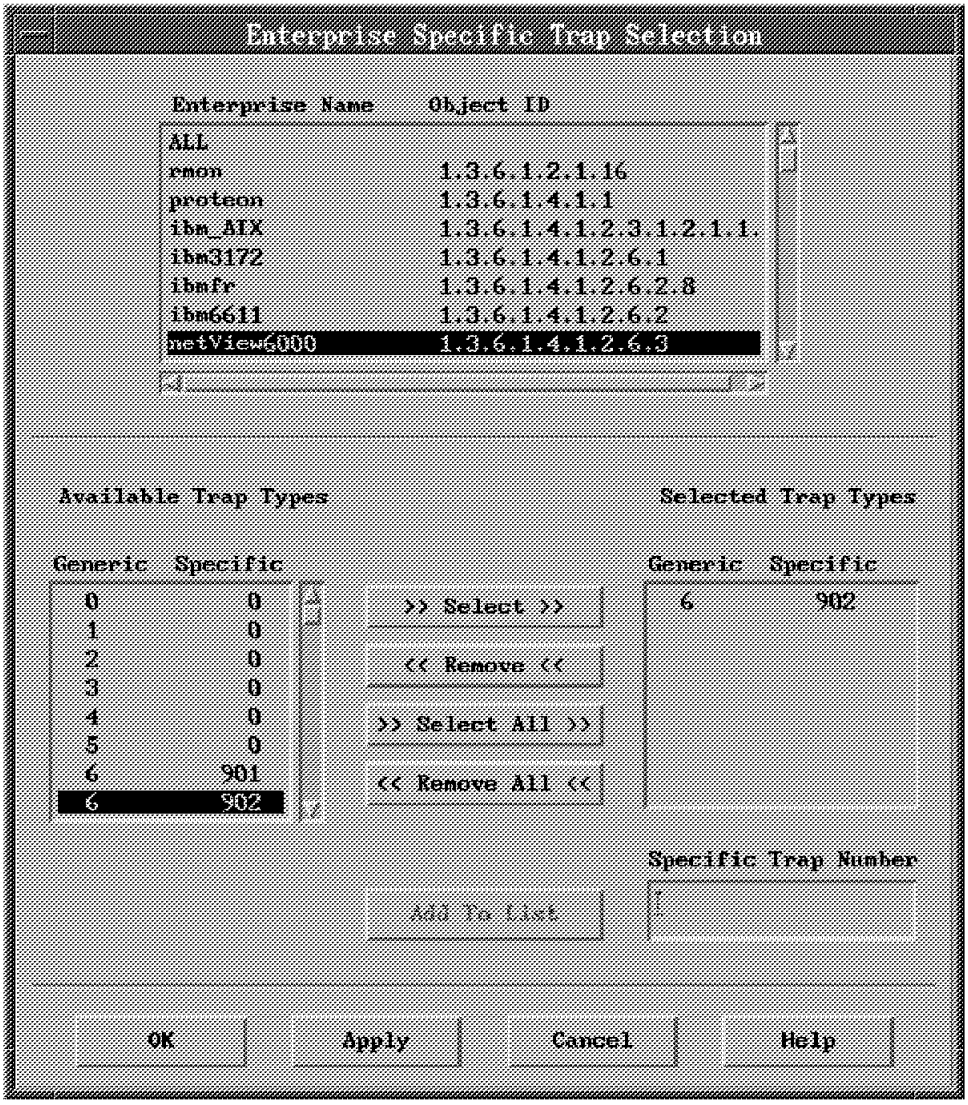


Figure 86. The Enterprise-Specific Trap Selection Window

- Select enterprise Name **netView6000**.
- Select Generic trap **6**.
- Select Specific trap **902**.
- Now press **Select, Apply** and **OK**.
- Select any specific hostnames or IP addresses required. In the example the machine rs60003 was added to the field **Name or IP Address**. This was done by selecting from **Objects equal to list**, typing in `rs60003.itso.ral.ibm.com` as the Name or IP Address, and then pressing **Add to list**.
- Select **OK**.

An example of the filter editor screen is shown in Figure 87 on page 124.

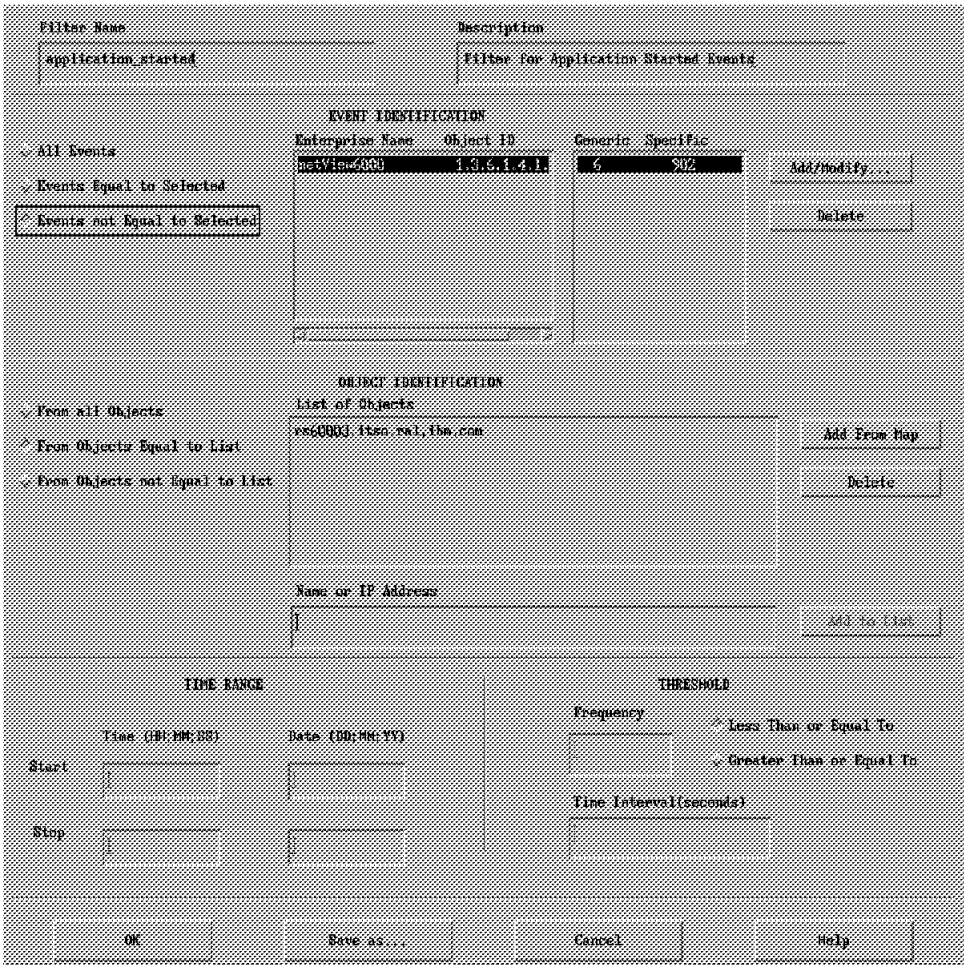


Figure 87. NetView for AIX Simple Filter Editor Screen

4.8.2 Activating a Filter

Once the filter has been defined, it needs to be activated for the current Events window. This is done as follows:

- Select **Options-> Filter Control** in the Events Display.
- Select the **application_started** filter from the available filters list.
- Select **Activate**.
- Select **Close**.

This can be seen in the screen in Figure 88 on page 125.

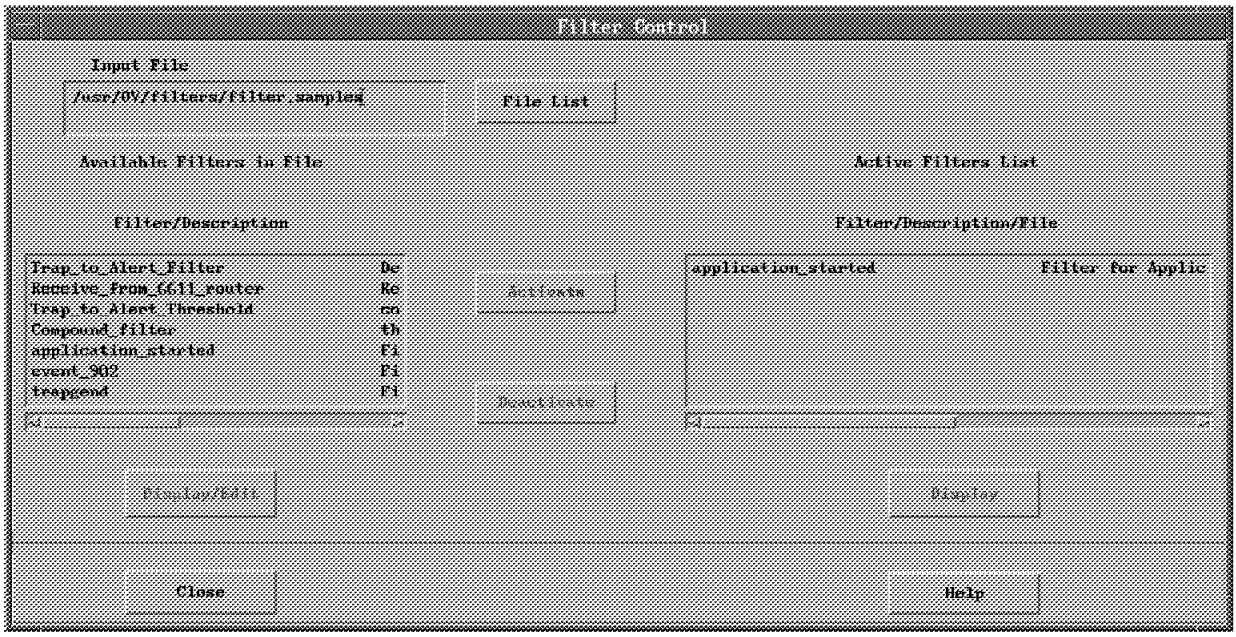


Figure 88. Filter Control Screen

The filter is now activated. To to test it, re-run the app_sendtrap shell script as discussed previously:

```
app_sendtrap rs60003 rs600010
```

You will notice that only the event defined for "Application Down" (901) will appear in this events window and event 902 will be filtered.

Note: This approach only affected this events window. It was not a global filter or, for example, did not affect the event 902 from being converted to an alert and being passed to S/390 NetView. Notice also that now events will be seen only from the node we selected; if the filter's rule for Event Identification *or* Object Identification (refer to Figure 87 on page 124) is *not* met, the event will be bypassed for the filtered nvevents/operator.

The active filter for the operator is kept in the operator's home directory, in for example:

```
/.root.events   (root = userid)

   /usr/OV/filters/filters.samples application_started
```

The above is used for each initiation of nvevents for the particular userid (root, in the above example).

If we wanted a more complex behavior (for example, to pass every trap except 902 from rs60002, plus any trap from any other node) we would have to combine simple filters together. It is possible to activate multiple simple filters or to combine simple filters together into a complex filter.

To de-activate the filter:

- Select **Options-> Filter Control**.
- Select **application_started** from the active filter list.
- Select **Deactivate**.

- Select **Close**.

The filter is now deactivated. To to test it, re-run the app_sendtrap and you will notice that both events, 901 and 902, will appear.

4.8.3 Using the Filter APIs

We have demonstrated the nvevents application using filters, but in fact filters can be used by any application using the event APIs.

The example that follows shows how a C program can register with the ovesmd daemon, and wait for events to arrive. When an event arrives, a shell script is executed to warn the user that the application trap 901 has arrived. This is useful if the NetView for AIX daemons are running but the GUI is not available. It also allows more selectivity than the automation afforded by the ″Command for Automatic Action″ option (see Figure 62 on page 95), since nodes or lists of nodes can be specified in the filter definition (see rs60003.itso.ral.ibm.com in ″List of Objects″ in Figure 87 on page 124).

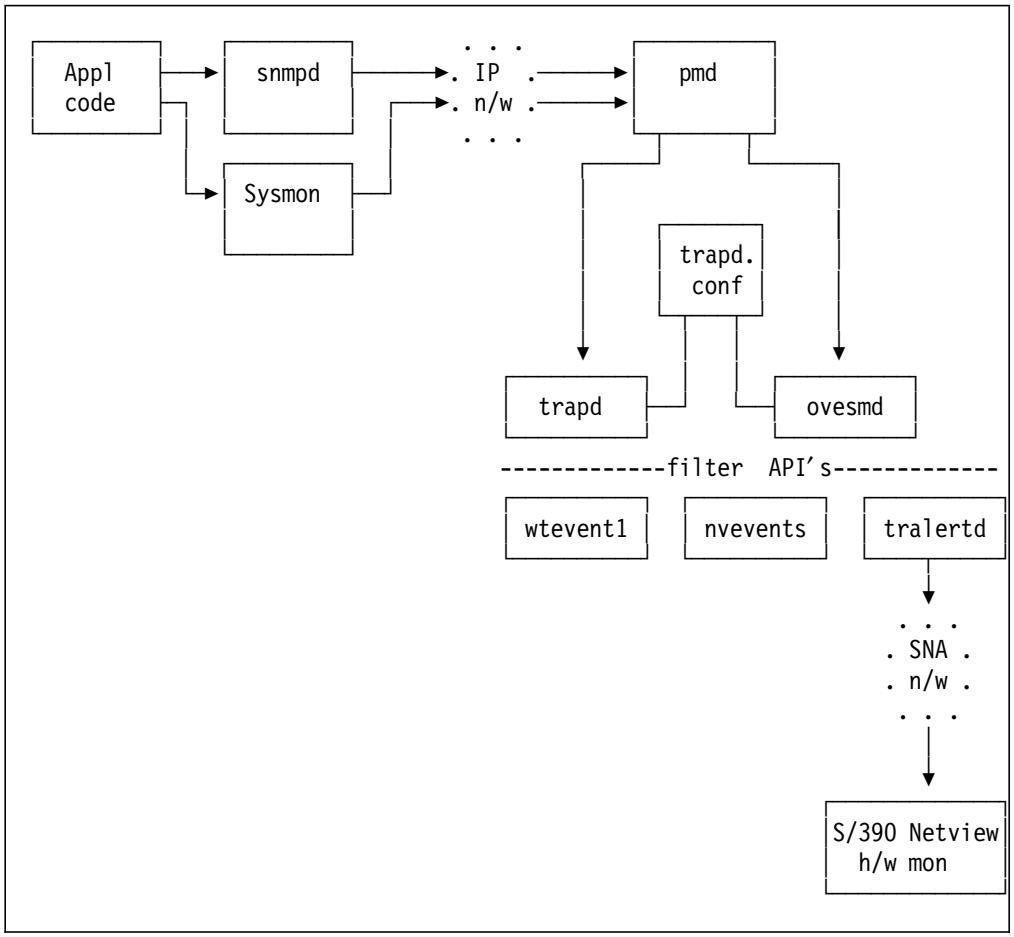


Figure 89. Configuration/Filtering Points for Events

The sample C program will register with ovesmd and wait for the event ″902″. When this event arrives a script will be executed to warn the operator. Create a new filter for event 902 as follows:

- Select **Tools> Filter Editor**.
- Select **Add Simple**.

- Enter the Filter Name: event_902.
- Enter the Description: Filter for event 902.
- Select **Events Equal to Selected**.
- Select **Add/Modify...**.

  The enterprise-specific trap selection menu now appears, as can be seen in Figure 86 on page 123.
- Select enterprise Name **netView6000**.
- Select Generic trap **6**.
- Select Specific trap **902**.
- Now press **Select, Apply** and **OK**.
- Select **OK**.

For this sample program see Figure 90.

```
/*********************************************************************/
/*********************************************************************/
/* Example Program to activate a filter and execute a command      */
/* on reception.                                                     */
/*                                                                   */
/* AUTHORs Rob Macgregor and Paul Fearn                              */
/* IBM UK                                                            */
/*********************************************************************/
/*********************************************************************/

#include <nvFilter.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <OV/OVsnmp.h>

/* Function for call back when trap arrives */
void    trap_arrived(OVsnmpSession *session,
                             OVsnmpPdu *trap, char * shell_cmd )
{
  system(shell_cmd) ;
}

int main(int argc, char **argv)
{
  int errcode,i ;
  OVsnmpSession   *snmp_session ;
  OVsnmpPdu       *trap_pdu ;
  struct  FilterNode *fptr ;
  char    filter_file[80] ;
  char    *filter_rule ;
  char    filter_desc[256] ;
  int     filter_rule_len=0 ;
  int     expand=1 ;

  if ( argc != 3 )
    {
      printf("Usage: wtevent1  <filtername> <command_to_run>\n") ;
      exit(-1) ;
    }

  /* Set up FilterNode structure */
  fptr = (struct FilterNode *) malloc(sizeof (struct FilterNode));
  fptr->FilterName = argv[1] ;
  fptr->FilterDescription = filter_desc ;

  strcpy(filter_file , "/usr/OV/filters/filter.samples") ;

  /* Find length of filter rule and allocate buffer to put filter text in*/
  errcode  = nvFilterGet(fptr , filter_file ,
                         NULL , &filter_rule_len , expand) ;

  filter_rule = malloc(filter_rule_len+1) ;
```

Figure 90 (Part 1 of 2). wtevent1.c Sample Program

```
  /* Get filter text */
  if ((errcode  = nvFilterGet(fptr , filter_file ,
                              filter_rule , &filter_rule_len , expand)) !=0 )
    {
        printf ("Cannot load filter rule %s\n" , fptr->FilterName) ;
        printf ("Reason - %s\n" , nvFilterErrorMsg(errcode) ) ;
        exit(2) ;
     }

/* Next we will wait for an event to pass the filter, invoking routine
   trap_arrived when a trap arrives           */

  printf("nvFilterGet got: %s\n", filter_rule);

  if ((snmp_session = nvSnmpTrapOpenFilter(NULL, NULL, filter_rule))
 == NULL) {
    perror("snmp session setup failed\n") ;
    printf("  errno=%d\n", errno);
    printf("  OVsnmpErrno=%d => %s\n",
                          OVsnmpErrno, OVsnmpErrString(OVsnmpErrno));
    printf("  nvSnmpErrno=%d => %s\n",
                          nvSnmpErrno, OVsnmpErrString(nvSnmpErrno));
    printf("  nvSnmpSubsys=%d\n", nvSnmpSubsys);
    exit(2) ;
  }

  while(1) {
    if ((trap_pdu = OVsnmpRecv( snmp_session )) == NULL) {
      printf("OVsnmpRecv failed. Error - %d\n" , OVsnmpErrno) ;
      exit(2) ;
    }

    trap_arrived(snmp_session, trap_pdu, argv[2]);
    OVsnmpFreePdu(trap_pdu) ;

  }

  sleep(3);
  OVsnmpClose(snmp_session) ;

  exit(0) ;
}
```

Figure 90 (Part 2 of 2). wtevent1.c Sample Program

To execute the program from an AIX window type:

```
wtevent1 event_902 "/usr/OV/bin/ovxbeep 902" &
```

where event_902 is the name of the filter, and ovxbeep is the command executed when this 902 event is received by the program.

The program will show, via printf:

```
 nvFilterGet got: ((CLASS=1.3.6.1.4.1.2.6.3 && (SNMP_SPECIFIC=902)))
```

To test this, re-run the app_sendtrap script as discussed previously:

```
app_sendtrap rs60003 rs600010
```

A red pop-up window displaying the number 902, signifying that this event has been received, will be displayed by the program.

Leave the program running and exit from the NetView for AIX EUI. Re-run app_sendtrap as above and the filter-program window will still appear. This example has given us a way to notify the user of a problem, or to initiate some automatic action, even when the NetView for AIX GUI is not available.

4.9 Dynamic Workspaces in NetView for AIX

NetView for AIX now allows for the definition of multiple dynamic workspaces. This is in addition to the static workspaces available in AIX NetView/6000 V2R1.

You can have a number of event workspaces open concurrently, all of which could have different options, such as the filters applied to the events in the workspace window. The following example shows two dynamic workspaces, with events selected on the following basis:

1. All events for machine rs60002 except 902s (application started)
2. All critical events from the network

4.9.1 Dynamic Workspace Creation for Example 1

Doing the following from the Events Display Application Main Workspace (the primary Control Desk menu):

- Select **Create-> Dynamic Workspace....**

This will result in the panel shown in Figure 91.

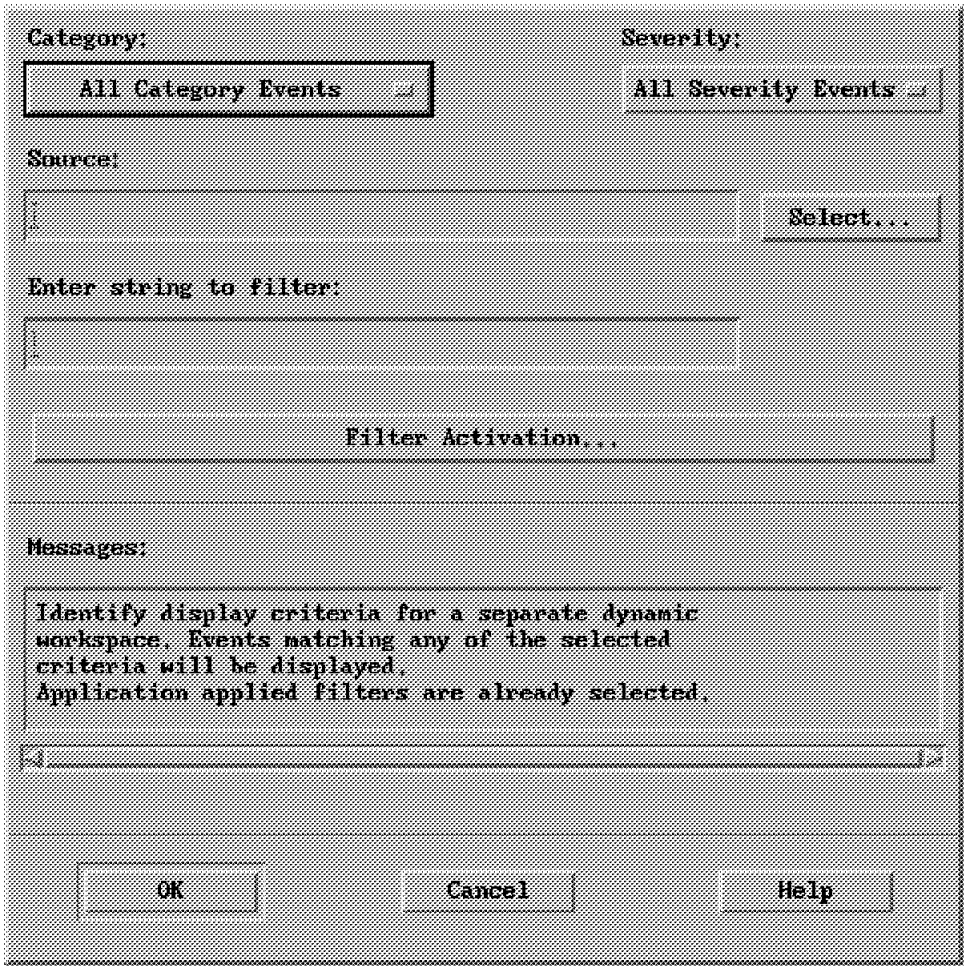


Figure 91. The Dynamic Workspace Panel

From the dynamic workspace panel, clicking on the Category button results in the following panel.

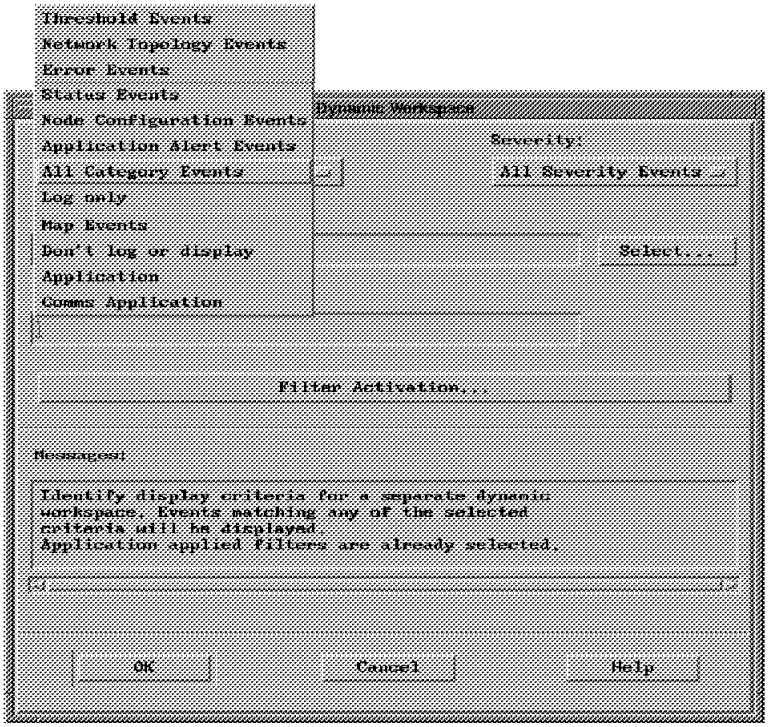


Figure 92. Selecting From Category in Dynamic Workspace Panel

Then, continuing on as follows will result in a dynamic workspace being created.

- Select **Status Events**.
- Select **Filter Activation**.
- Select **application_started**.
- Select **Activate**.
- Select **Close**.
- Select **OK**.

A dynamic workspace will now be created, as shown in Figure 93 on page 131.

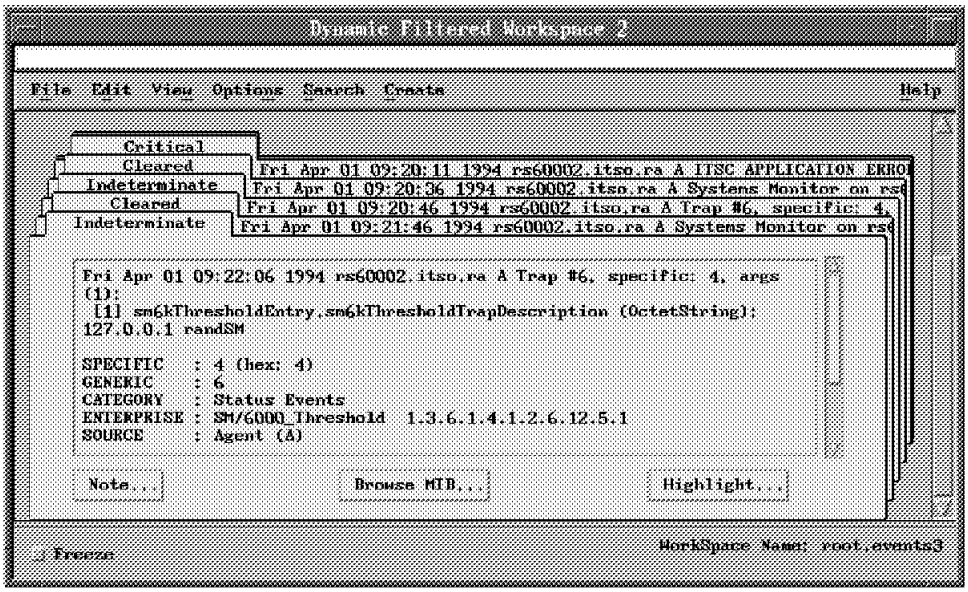


Figure 93. Dynamic Workspace Example 1

This shows the Status Events for rs60002 as requested in the filter ″application_started″. In the panel presented as in Figure 91 on page 129, you could qualify the selected events by entering something in the ″Enter string to filter:″ field, such as: ITS0. This would have shown only events which contained the text: ITSO.

4.9.2 Dynamic Workspace Creation for Example 2

Do the following from the Events Display Application Main Workspace (the primary Control Desk menu):

- Select **Create->Dynamic Workspace....**

  You will then see the panel as in Figure 91 on page 129.

- Select **Critical**
- Select **OK**

This will display all critical events from managed nodes in the network.

Other examples might include selecting events from a particular source. All netmon events can be selected by **Netmon (N)** from the Select Event Source panel displayed by selecting **Dynamic Workspace-> Select...**.

To toggle between the workspaces, click on the **Events** icons located in the Control Desk window.

The dynamic workspaces can be relocated by clicking the middle mouse button on the title portion of the event window, and then using drag and drop.

The following two figures indicate the status of two different workspaces and indicate the status of filters for each workspace.

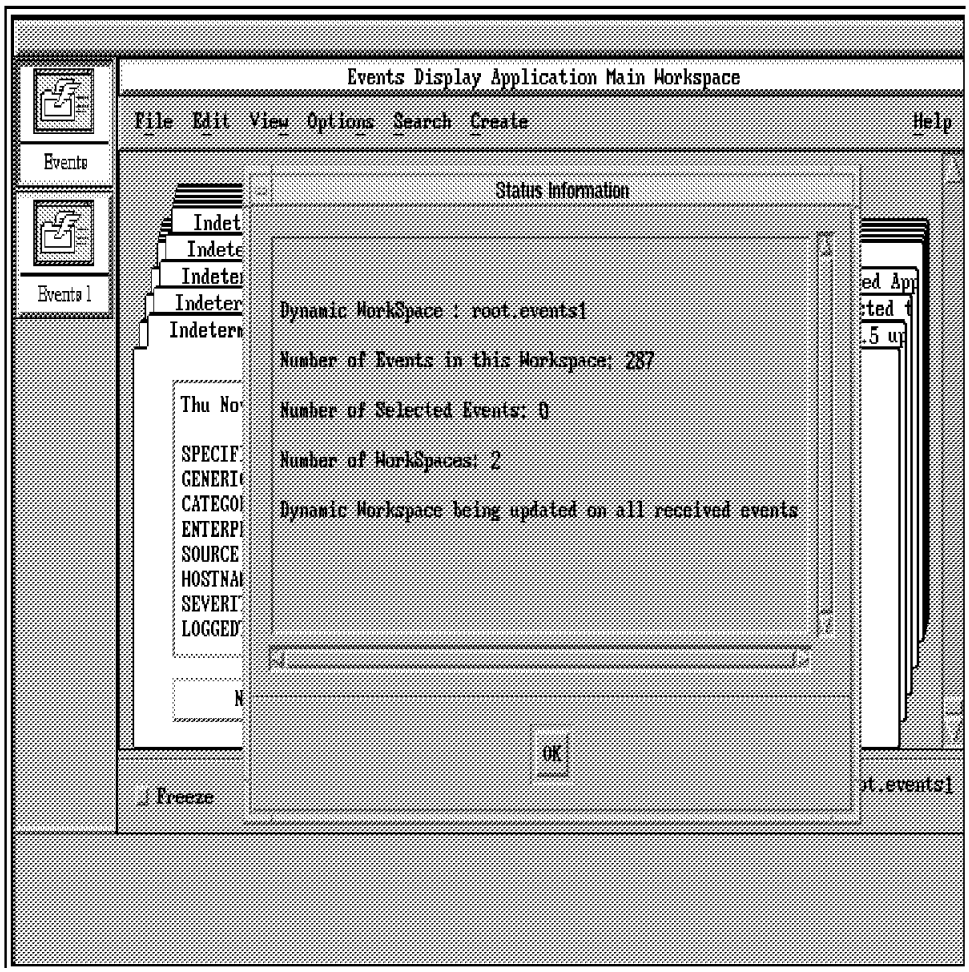


Figure 94. Options -> Show Status root.events1. No active filters are shown as in progress.

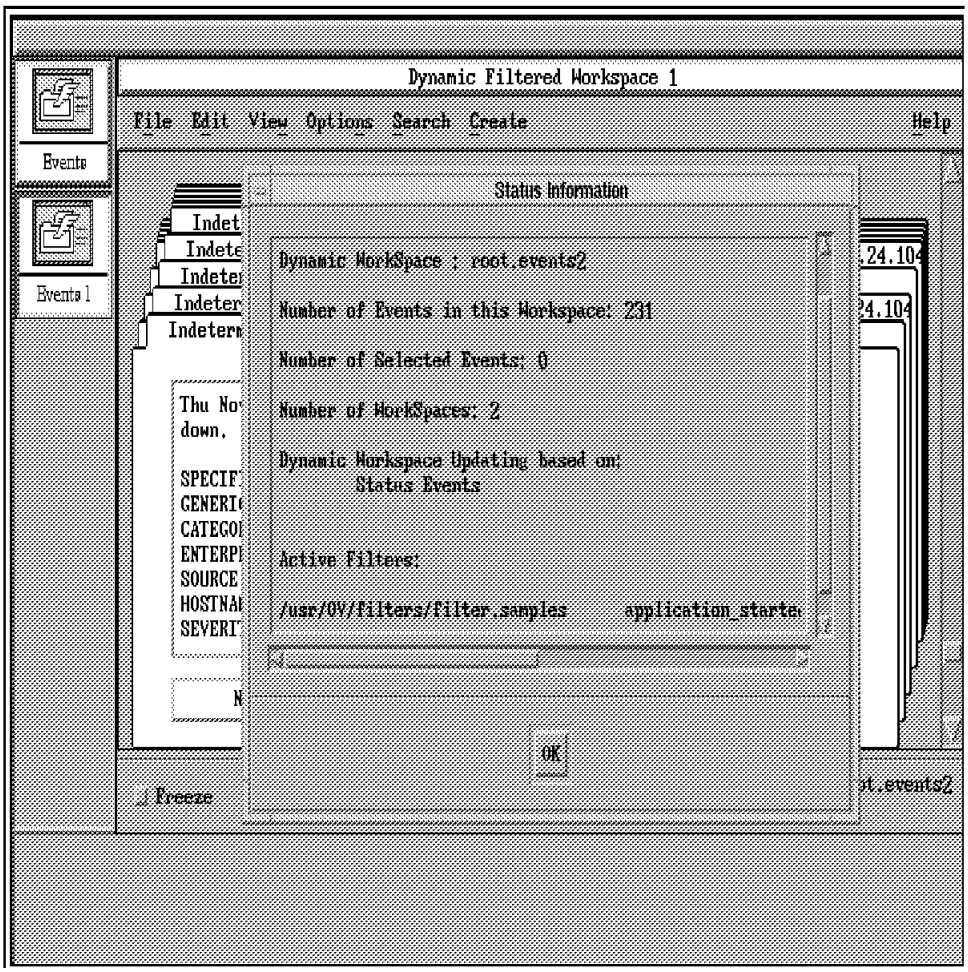


Figure 95. Options -> Show Status root.events2. Indicating application_started filter is active.

4.9.3 Searching for Events in the Current Event Workspace

It is possible with NetView for AIX to search by criteria or to search by filter.

4.9.4 Searching by Criteria

From an events Workspace:

- Select **Search**.
- Select **By Criteria**.

  You will then see the panel as displayed in Figure 96 on page 134.

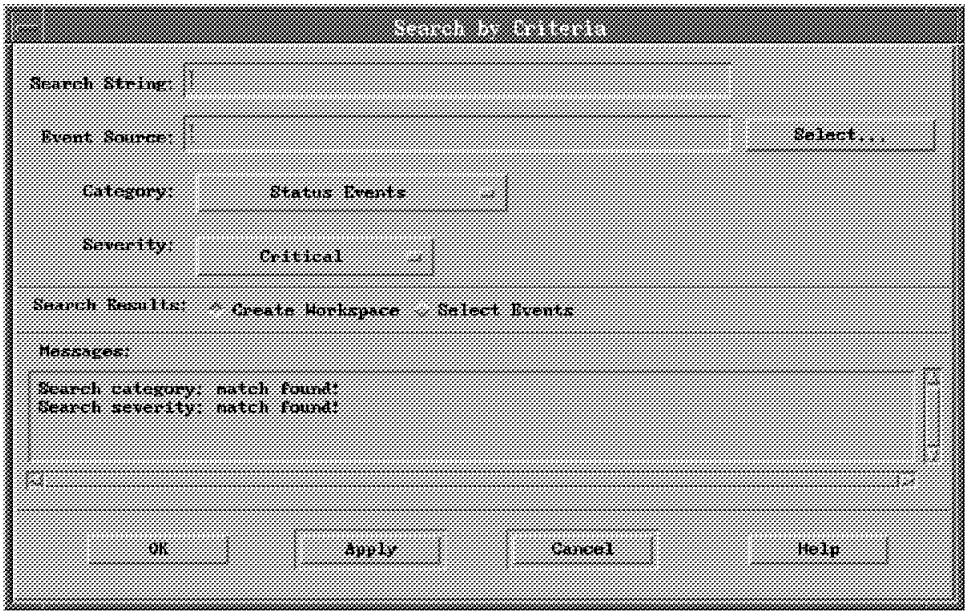


Figure 96. The Search by Criteria Window

- Select **Category**.
- Select **Status Events**.
- Select **Severity**.
- Select **Critical**.
- Select **Create Workspace**.
- Select **Apply**.

A static workspace is now created, as in the example in Figure 97 on page 134.

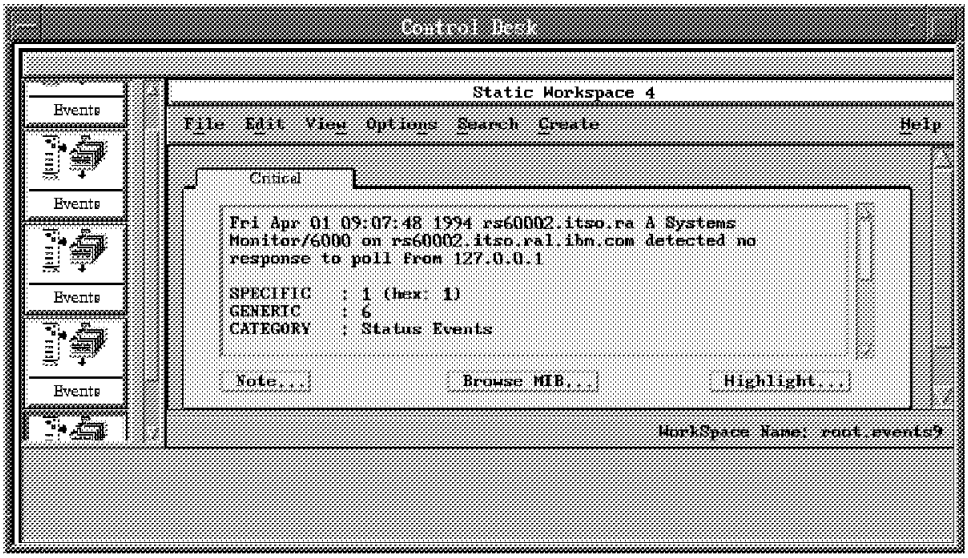


Figure 97. A Static Window Created with a Search by Criteria

4.9.5 Searching by Filter

From an events workspace:

- Select **Search**.
- Select **By Filter**.
- Select **application_started**.
- Select **Activate**.

  This process will highlight all the events cards that have passed this filter. You can then create a static workspace, containing these selected cards. To do this:

  1. Select **Create**.
  2. Select **Static Workspace**.
  3. Select **Selected**.

  This then creates the static workspace.

4.10 Displaying Event Information

NetView for AIX will sometimes receive a large number of events generated from the managed network. While viewing the events a number of pre-defined actions can be executed; for example, sorting and printing events.

The following example shows how to display the number of SNA events generated and sorted by nodename.

To define such an example, do the following:

- Select **Options->Event Configuration->Trap Cust..** from the main NetView for AIX screen.
- Select **Configure Additional Actions...**
- Click on **Clear Fields**.
- Enter `Display ITSO SNA Event Statistics` in the Title for Action field.
- Enter `rae_oper.sh` in the command field. See Figure 100 on page 138.
- Enter `20s` for the maximum time to wait.
- Enter `Show Specific ITSO Events Relating to SNA App..` in the Description field.

  See Figure 98 on page 136.
- Click on **Apply** followed by **Cancel**.
- Click on **Apply** on the Event Configuration window.

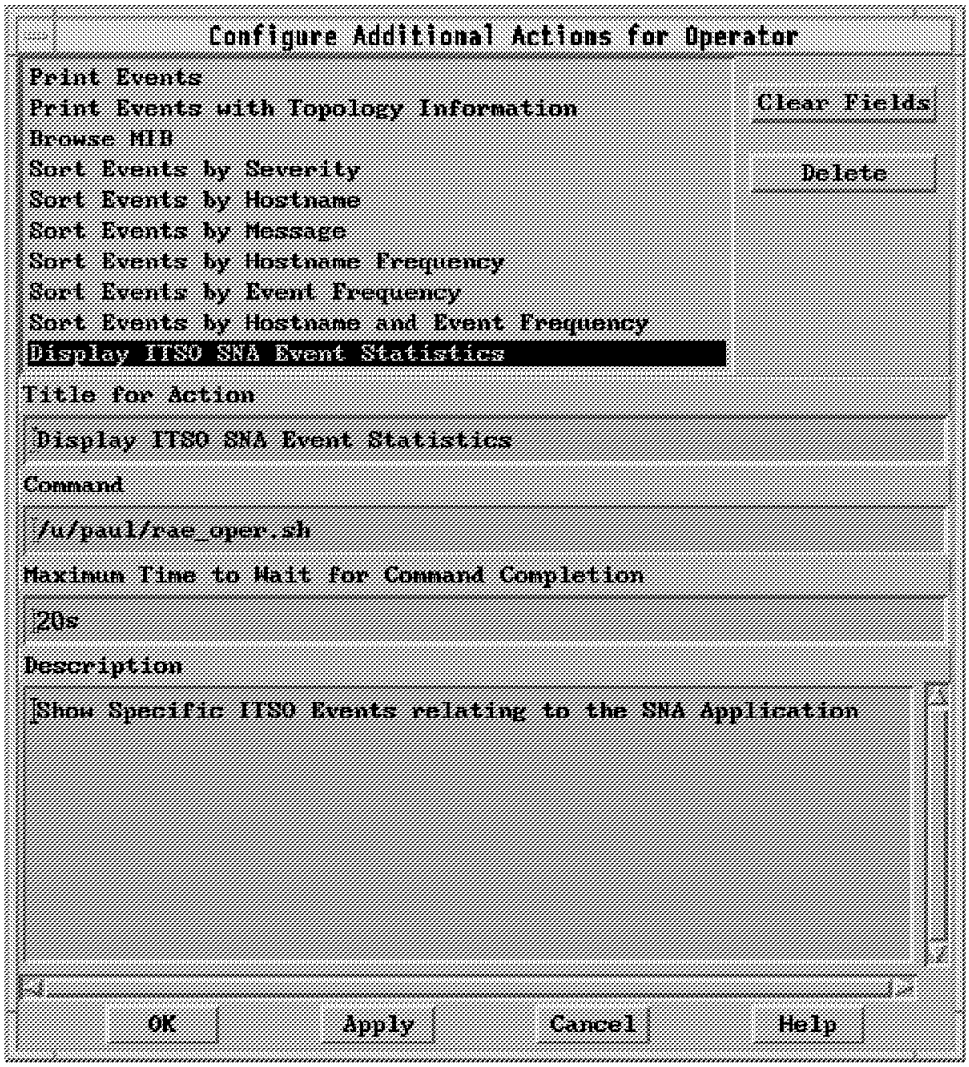


Figure 98. Configure Additional Actions for Operator

The new operator option is now defined. To test the new option: from the events display window select from within the Control Desk:

- **Options->Additional Actions->All**.
- Select **Display ITSO SNA Event Statistics** from the available actions list.
- Select **Apply**. This will display the results similar to screen shown in Figure 99 on page 137.
- Click on **Close** when complete.

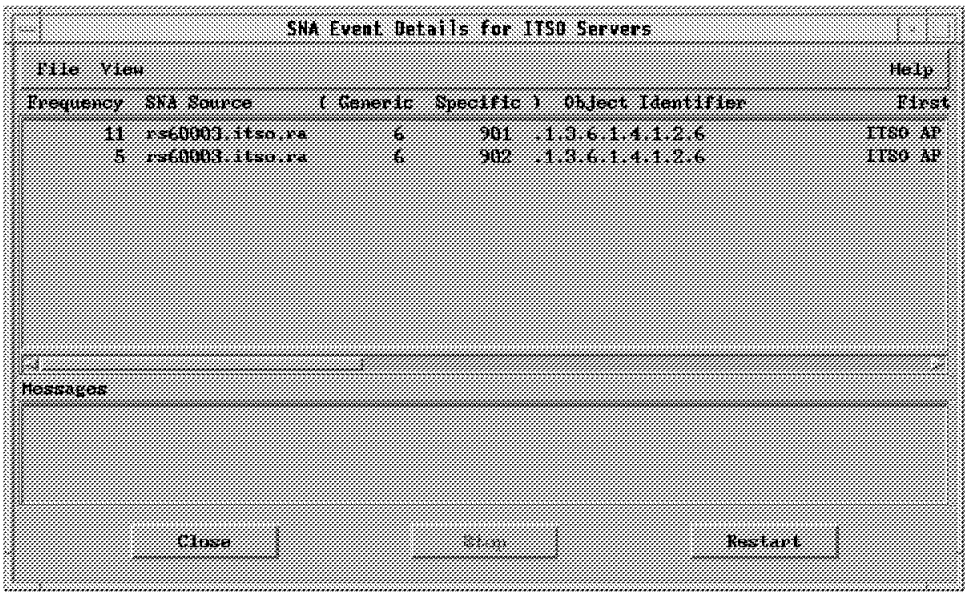


Figure 99. The Output from the Calculation of SNA Events

The sample script has been modified from the existing NetView for AIX command `freqSortEnvt` to show how to display specific types of events for specific machines.

For this sample script see Figure 100 on page 138.

```
#!/bin/sh
##################################################
# Script Based on the NetView for AIX
# /usr/OV/bin/freqSortEvnt script to show the example
# Modified By P. Fearn
##################################################

#
NAME=`basename $0`
TMP=/tmp/${NAME}$$              # Temporary file
TITLE="SNA Event Details for ITSO Servers"
SORTCOLUMN2=9
AWK=awk
set -e                  # exit if we encounter an error

# Begin Awk Script

$AWK 'BEGIN{
    num=0
    # create heading line
    SOURCEWIDTH=25
    OIDWIDTH=30

                        printf("Frequency")
                        printf("  SNA Source    ")
                        printf("  ( Generic  Specific )  Object Identifier First Seen Message\n)
}
{
    #
    # make a unique "key" depending on how we are to sort
    #
    OID = $1
    GEN = $2
    SPEC = $3
    SRC = $10
    if ("'"$1"'" == "-s")
                        KEY=SRC # source field is key
    else {
                            NODESTR=sprintf("%-15.15s  ", SRC) # include source in key
                        # Build long key string
                        KEY=sprintf("%s%7d  %8d  %-30s", NODESTR, GEN, SPEC, OID);
    }
    if (freqArray[KEY] == 0) {
                        if ("'"$1"'" != "-s") {
                            for (i=1; i<=10; i++)
                         $i=""
                            sub("  *", "")
                            sampleMsgArray·num“ = $0
                        }
                        keyArray·num++“ = KEY  # Have not seen this keyword before
    }
    freqArray·KEY“++
}
END {
    for (i=0; i<num; i++) {
                        key = keyArray·i“
                        printf("%9d  %s", freqArray·key“, key)
                        if ("'"$1"'" == "-s")
                            printf("\n");
                        else
                            printf(" %s\n", sampleMsgArray·i“);
    }
}' > $TMP
#
# We succeeded.  Run the command in the background so we can return successfully
#

(/usr/OV/bin/xnmappmon -commandTitle "$TITLE" -commandHeading "$HEADING" \
    -geometry 700x400 -sortColumn1 1 -sortColumn2 9 \
    -headingLine 1 -sort -reverseSort -cmd grep SNA $TMP ) &
```

Figure 100. rae_oper.sh Script

4.11 Event Log

All the events that are received are logged in the file /usr/OV/log/ovevent.log.

This file is regularly monitored for its size, as it will grow quite quickly depending on the number of events being raised. NetView for AIX will archive the log file /usr/OV/log/ovevent.log when it reaches a maximum number of bytes which can be modified by the user. The archive file is called /usr/OV/log/ovevent.log.BAK.. We suggest that you use CRON to journal this archive file.

You can change the size of the ovevent.log file, by selecting the following options from the Events History Application Main Workspace pull-down menus. Events History is accessed from the main Root pull-down or by dragging Events History from the Tools window.

- Select **Monitor-> Events-> Event History**.

  This displays the Event History window.

- Select **Options->Set Log Size**.

  Modify the field New Maximum Event Log Size in Kb to the required value up to a maximum of 2MB.

- Select **OK**.

4.12 Trap to Alert Conversion

This chapter details the requirements for sending events to S/390 NetView. In addition to NetView for AIX, the AIX Service Point application and SNA services are required to achieve this.

In our sample environment, we used an LU6.2 SNA connection from the RISC System/6000 to the MVS/ESA host. This provided the transport services required to exchange status and commands between the hosts.

With this connection, it is possible to:

1. Send NetView for AIX events to S/390 NetView.
2. Send messages, using (for example) a user-written routine: nvsendmsg from the RISC System/6000 to a S/390 NetView operator:

   ```
   nvsendmsg S/390 NetView_operator_id {'message text'}
   ```

3. Send commands from S/390 NetView to execute on the RISC System/6000.

When NetView for AIX is involved, the daemons spappld and tralertd are required to be running, as are the AIX NetView Service Point daemons.

To check that the required daemons are running, use SMIT or enter the commands:

```
ovstatus tralertd
ovstatus spappld
/usr/lpp/nvix/scripts/nvix_control status
```

Refer to the *AIX NetView Service Point Installation, Operation and Programming Guide,* SC31-6120 and *IBM NetView for AIX and the Host Connection,* SC31-6235 for additional information.

Messages and commands are carried to and from the S/390 NetView host in a structured format, using Network Management Vector Transport or NMVTs. NMVTs are made up of subvectors, and the different subvectors have different uses.

NMVTs that carry events from the RISC System/6000 to S/390 NetView, can be viewed by a S/390 NetView operator using the hardware monitor (NPDA).

The details of the NetView for AIX event as it passes to S/390 NetView, are carried in Code Points within the message NMVT. These code point numbers are translated back into meaningful text by the S/390 NetView hardware monitor.

In the remainder of this chapter, the term: *alert* is meant as the NMVT which results from trap-to-alert conversion from NetView for AIX.

If an AIX system is configured to send alerts to the main frame, it is called a *service point*.

The flow of NMVTs can be supported by SNA SSCP-PU or LU6.2.

Commands are sent from a S/390 NetView operator or exec, via the RUNCMD SPCS command. This generates a command NMVT which is received by the NetView for AIX spappld daemon, which then sends the command to the service point application that is defined in its daemon configuration.

4.13 Host Interaction Examples

In these examples we will:

1. Send a NetView for AIX event to S/390 NetView.
2. Have S/390 NetView respond to a NetView for AIX event.
3. Send some mainframe status to NetView for AIX.
4. Have NetView for AIX respond to a S/390 NetView event.

This will illustrate how S/390 NetView and NetView for AIX can be either the manager or the agent.

4.13.1 Connection with RISC System/6000 Service Point and S/390 NetView

To use AIX NetView Service Point in conjunction with S/390 NetView, it is necessary to configure AIX SNA profiles which can be used in conjunction with S/390 definitions.

Appendix G, “Selected AIX SNA Server Profiles” on page 291 contains sample AIX SNA Server/6000 definitions which were used for this project′s LU 6.2 connection.

4.13.2 Sending a NetView for AIX Event to S/390 NetView

Earlier, we discussed the event: 901. This event, when converted by NetView for AIX into an NMVT(alert) from a trap, is uncustomized; that is, an alert with default information attached.

We use app_sendtrap as discussed previously and when NetView for AIX processes the trap, the trap is converted to an NMVT/alert and as a result of

this, we see the following alert appear in S/390 NetView hardware monitor, as in Figure 101 on page 141.

```
 N E T V I E W          SESSION DOMAIN: RAPAN    WTWKSH6   11/17/94 14:31:15
 NPDA-43S                      * EVENT DETAIL *                 PAGE  1 OF  3

  RAPAN       RA6003CP     RS60003T     RS60003      RS60003
             +--------+   +--------+   +--------+   +--------+
  DOMAIN     |  SP    |---|  TP    |---|  DEV   |---|  DEV   |
             +--------+   +--------+   +--------+   +--------+

 DATE/TIME: RECORDED - 11/17 14:30    CREATED - 11/17/94 14:29:19

 EVENT TYPE: UNKNOWN

 DESCRIPTION: SNMP RESOURCE PROBLEM

 PROBABLE CAUSES:
    UNDETERMINED




 ENTER A (ACTION) OR DM (DETAIL MENU)

 ???
CMD==>
```

Figure 101. The S/390 NetView View of an Uncustomized SNMP Alert

Looking at the event in more detail, we can see what is shown in Figure 102.

```
 N E T V I E W          SESSION DOMAIN: RAPAN    WTWKSH6   11/17/94 14:31:25
 NPDA-43S                      * EVENT DETAIL *                 PAGES 2 and 3

  RAPAN       RA6003CP     RS60003T     RS60003      RS60003
             +--------+   +--------+   +--------+   +--------+
  DOMAIN     |  SP    |---|  TP    |---|  DEV   |---|  DEV   |
             +--------+   +--------+   +--------+   +--------+

 QUALIFIERS:
    1)  LOG ID 2ECBAF0F00000000
    2)  ENTERPRISE netView6000
    3)  SNMP GENERIC-TRAP NUMBER ENTERPRISE SPECIFIC
    4)  SNMP GENERIC-TRAP NUMBER 902
    5)  SNMP MIB VARIABLE NAME .1.3.6.1.2.1.1.1
    6)  SNMP MIB VARIABLE VALUE Daemon Restored
    7)  SNMP MIB VARIABLE NAME .1.3.6.1.2.1.1.1
    8)  SNMP MIB VARIABLE VALUE rs600010.itso.ral.ibm.com
    9)  SNMP MIB VARIABLE NAME .1.3.6.1.2.1.1.1

 UNIQUE ALERT IDENTIFIER: PRODUCT ID - 5696-7310  ALERT ID - DDB3159A

 ENTER A (ACTION) OR DM (DETAIL MENU)

 ???
CMD==>
```

Figure 102. Alert Detail with No Customization

4.13.3 Customizing NetView for AIX Aimed at S/390 Host Alerts

In this section, we will discuss briefly ″code points″. Code points are used within an NMVT to assist S/390 operators and software to better understand what information has arrived from a service point. It is imperative that the service point and S/390 are in agreement with regards to code points and their meaning. See your S/390 programmer for details on this subject.

We will add some useful information to the alert resulting from app_sendtrap, by changing the event to add some NetView for AIX code point definitions.

These code point definitions cover the following types of information.

Table 12. NetView for AIX Code Points

Description	SubVector	errmsg Set ID
Detail Data	x'98'	D
Error Description	x'92'	E
Failure Cause	x'96'	F
Install Cause	x'95'	I
Probable Cause	x'93'	P
Recommended Action	x'81' 1	R
User Cause	x'94'	U

Notes.

1 is a subfield and not a subvector. See *Systems Network Architecture Formats,* GA27-3136 for additional information.

The code point definitions have to be added to the AIX error message catalog before we can use them in NetView for AIX. This catalog is called `/usr/adm/ras/codepoint.cat`.

You can use the AIX errmsg command to display the currently configured code points. Choose a value from the errmsg Set ID column in Table 12 to display information for a particular group of code points. For example: `errmsg -w D`

To add new code point definitions to the catalog, we did the following:

- Created a file called:

  `ITSO_codepoints`

- Entered the details as shown in Figure 103 on page 143.

```
* Sample Configuration File
* Use with the AIX errinstall command
* for Adding New codepoint definitions
*
* SET D - Detailed Data          S/390 subvector: x'98'
*     E - Error Description      S/390 subvector: x'92'
*     F - Failure Cause          S/390 subvector: x'96'
*     I - Install Cause          S/390 subvector: x'95'
*     P - Probable Cause         S/390 subvector: x'93'
*     R - Recommended Action     S/390 subfield:  x'81'
*     U - User Cause             S/390 subvector: x'94'
*
*
* Message ID Must Be 4 characters and in hex format
* The Message TEXT must not exceed 40 characters
*
* Start the additional user defined codepoints from E601 - E999
*
SET D
E601 "ITSO D APPLICATION FAILURE"
E610 "ITSO D APPLICATION RESTORED"
SET E
E601 "ITSO E SNA DOWN"
E610 "ITSO E SNA NOW AVAILABLE"
SET F
E601 "ITSO F SOFTWARE DOWN"
E610 "ITSO F SOFTWARE HAS RESTARTED"
SET I
E601 "ITSO I MEMORY USED"
E610 "ITSO I MEMORY FREE"
SET P
E601 "ITSO P MEMORY2"
E610 "ITSO P MEMORY"
SET R
E601 "ITSO R RESTART APPLICATION"
E610 "ITSO R MONITOR APPLICATION"
SET U
E601 "ITSO U APPLICATION DOWN"
E610 "ITSO U APPLICATION UP"
```

Figure 103. ITSO_Codepoints

- Type `errinstall -c ITSO_codepoints` to check for syntax errors.
- Type the command `errinstall -f ITSO_codepoints` to add these new code points.

  The "-f" option will replace any duplicate entries.

You can now check that these code points are available by typing:

```
errmsg -w ALL | grep E601
errmsg -w ALL | grep E610
```

If you find any incorrect descriptions, then you can delete them as follows:

1. Type errmsg
2. Type SET P
3. Type - E601 or - E610

4. Type <CNTRL>d

Note that there is one code point catalog for each language. The process listed above has updated /usr/adm/ras/codepoint.cat.

Using the command:

```
ln -s /usr/lib/nls/msg/En_US/codepoint.cat /usr/adm/ras/codepoint.cat
```

we linked the two files together and ensured that NetView for AIX was looking at the file we updated.

Now that we have defined the code points, we must associate them with the specific events. To do, do the following from the main NetView for AIX pull-down menu:

- Select **Options-> Event Configuration-> Trap Customization**.
- Select the **netView6000** Enterprise Name.
- Select the generic/specific event **6/902** (appl restarted).
- Select **Alert Editor...**

This will give you a screen as in Figure 104.

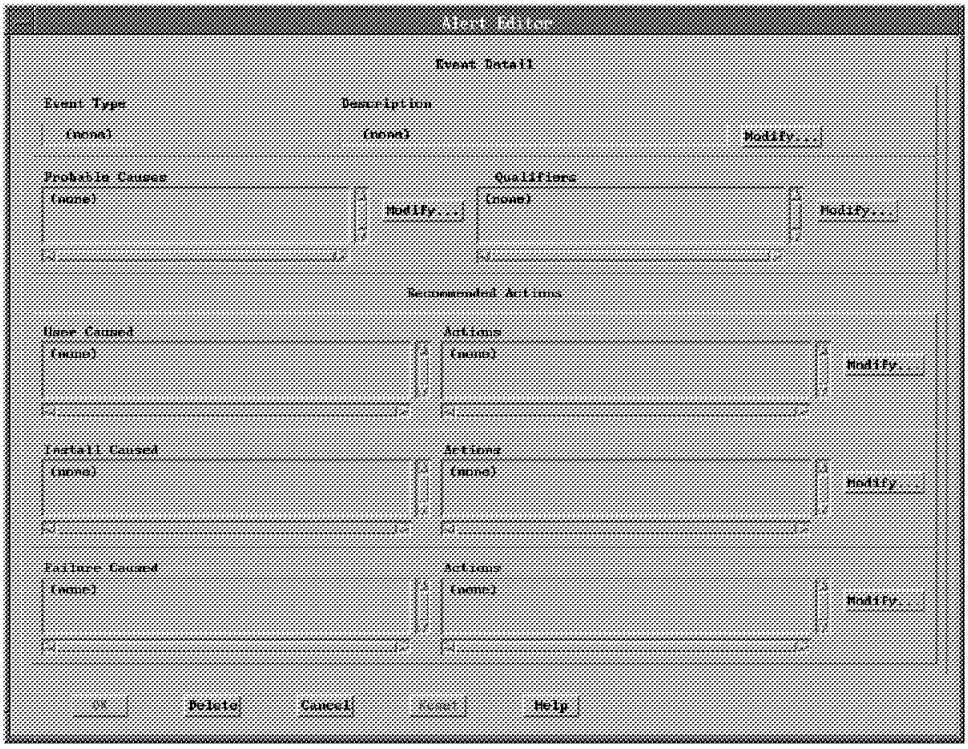


Figure 104. Alert Editor Primary Screen

4.13.4 Changing the Description Code Point

- Select the **Modify** next to the description field.

  This then displays the generic alert window.
- Select **Search** and type E610.
- Select **OK**.
- Cancel the search window, and select the Permanent radio button.

This then takes you to the screen as in Figure 105 on page 146.

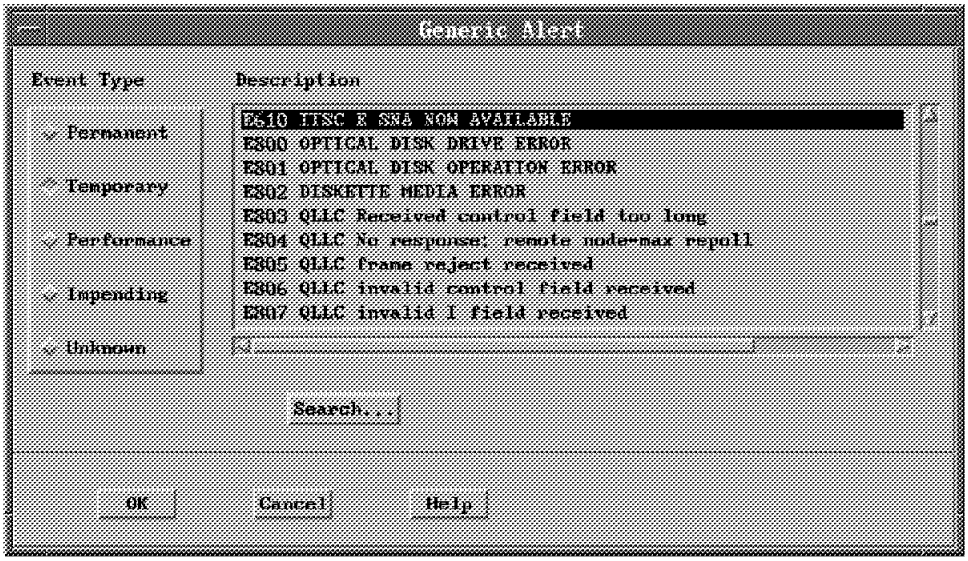


Figure 105. Generic Alert Window

Select **OK**.

4.13.5 Changing the Probable Cause Code Point

Do the following from the Alert editor window.

- Select **Modify** next to the Probable Causes Window.
- Select **Search** and type E610.
- Select **OK** followed by **Cancel**.
- Move the required code point from the Available probable Causes window to the Selected Probable Causes window, using the arrows (see Figure 106).

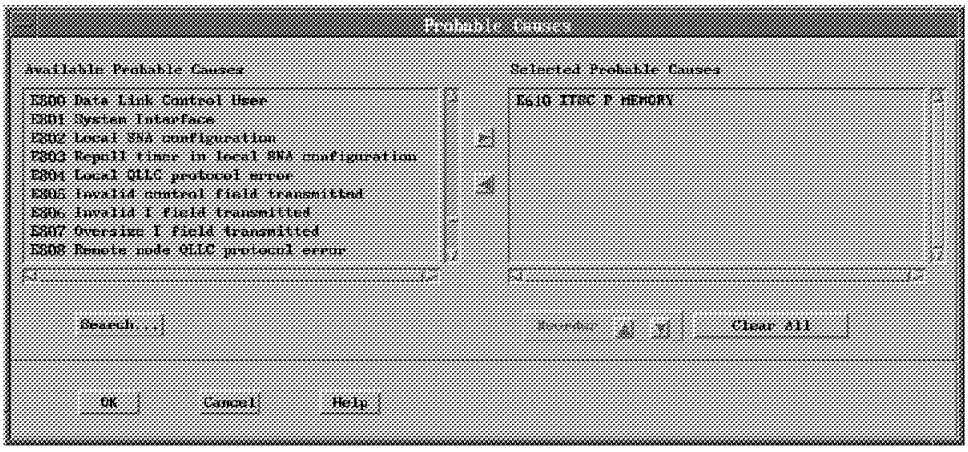


Figure 106. Editing Alert Probable Causes

- Select **OK**.

Repeat this process for all the other code point options.

Note: It is necessary to modify all fields in this panel if you started out with (none) within the field's contents. (none) will result in a general/default set of

code points being used and, repeating: It is necessary to modify *all* fields in such a case.

4.13.6 Code Point Qualifiers

It is possible to send variable data in the code point, using the variables available to you during event customization (refer to page 94).

To add a code point qualifier, select **Modify** from near the Qualifiers Window.

- Select **Add**.
- Choose **0036 Location** from the available list. See Figure 107.

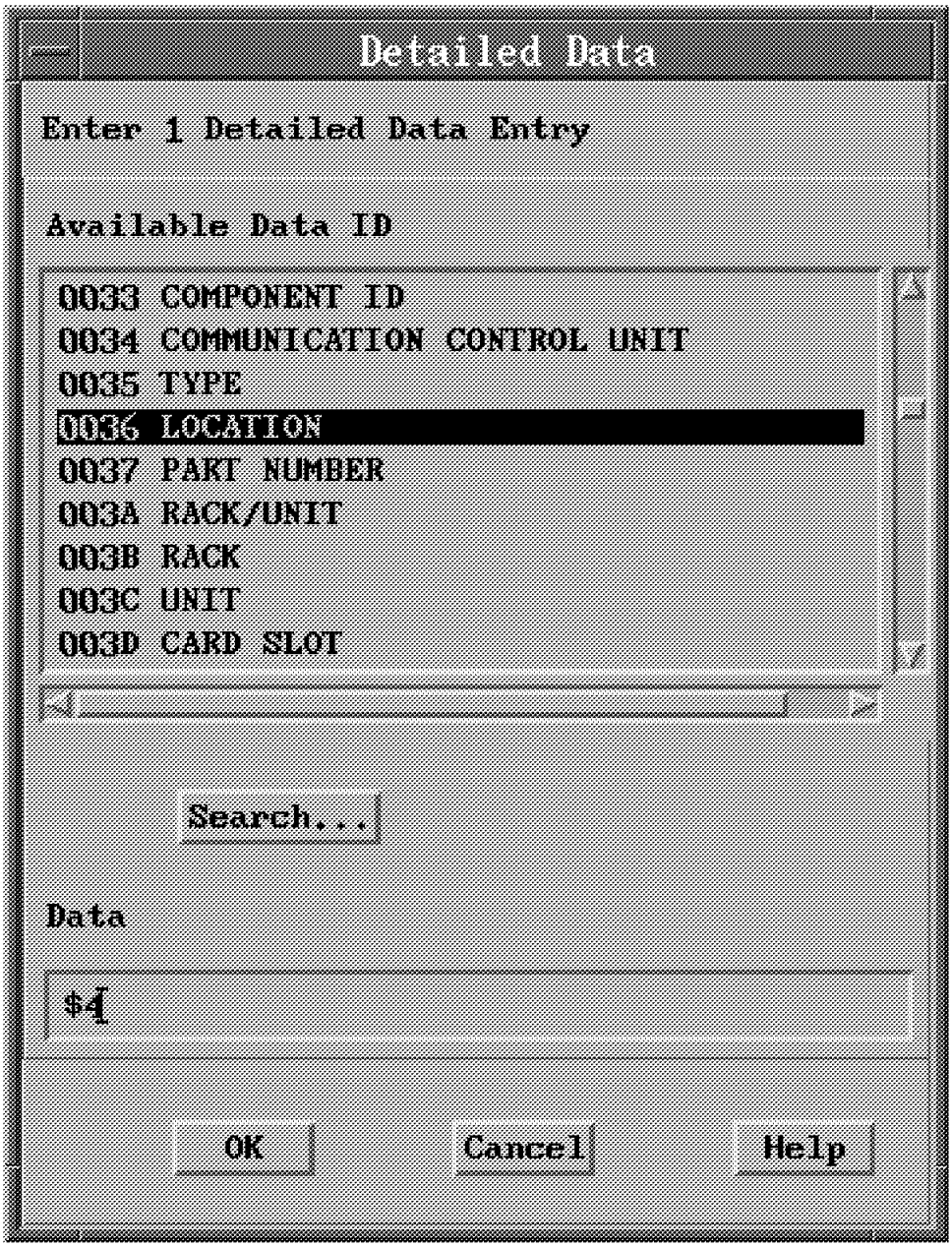


Figure 107. The Available Qualifiers List

- Enter $4 in the Data field.

- Select **OK**. which returns you to the Qualifiers window as in Figure 108 on page 148.
- Select **OK**.

Finally, this leaves us with the window as in Figure 109, so select **OK** to apply these changes.

Figure 108. The Qualifiers Window

The completed alert editor window is shown in Figure 109.

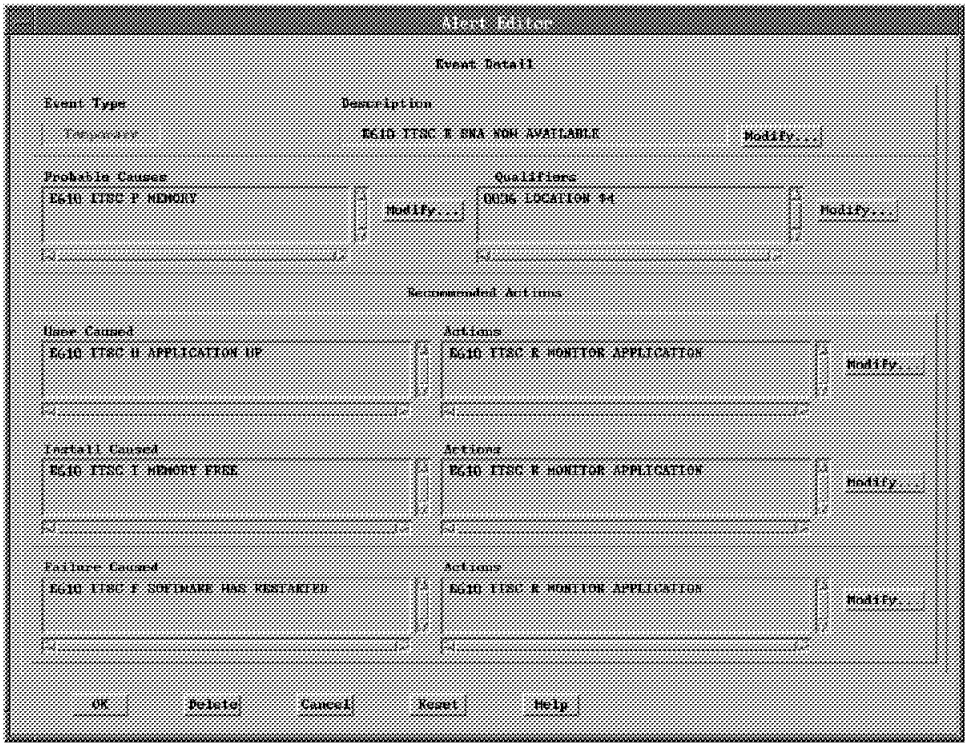


Figure 109. The Completed Event Window

Select **OK** or **Apply** and **Cancel** on the Event Configuration window.

4.13.7 Checking in S/390 NetView

This document does not intend to summarize all aspects of S/390 NetView and its handling of the configured RISC System/6000 service point generated NMVT (alert). Ensure you are in contact with a person who is familiar with S/390 NetView for details and to ensure you are in sync (the same values).

Some matters which will need discussions between the service point person and the S/390 NetView person include:

- Code points. They must be in sync at both ends of the connection.
- S/390 NetView filters. Although the service point NMVT may be flowing from the RISC System/6000 to the S/390, it may be filtered at the S/390 NetView end of the connection.

  For example, to see the status of some S/390 filters, from a S/390 NetView panel issue:

  ```
  npda df oper

    or

  npda df arec
  ```

- Service point application name. It is necessary for S/390 NetView to have access to the configured spappld application name.
- Service point LU name. This may seem obvious to both the S/390 NetView and the RISC System/6000 service point coordinators, but ensuring both parties are in agreement in this regard is worthy of discussion.

With the additional flexibility of LU 6.2 connections and the emergence of applications that make use of service point connections, the above matters must be discussed by persons responsible for installation and configuration of S/390 and RISC System/6000 when the NetView for AIX family of products is involved in network and systems management activities.

4.13.8 Default Trap to Alert Conversions

There are some events which are converted into alerts by default. These are inactive when NetView for AIX is initially installed.

The following example shows how to do the following:

1. View the list events in the default trap to alert filter.
2. Activate the trap to alert filters.
3. View the default code point definition for the Node Down event.

From the main NetView for AIX menus:

- Select **Options> Event Configuration**.
- Select **Start Filter Editor**.
- Select **Trap to Alert Filter control**.
- Select the **Trap_to_Alert_Filter** from the list.
- Select **Events Equal to Selected**.

- Select **Add/Modify**.
- Select enterprise **netView6000** from the event ID list.

We can now browse the list of predefined generic/specific event pairs from the Generic Specific list. To display more information about any of these traps, use the

```
event -l
```

command.

See Figure 110 for the list browsing screen.

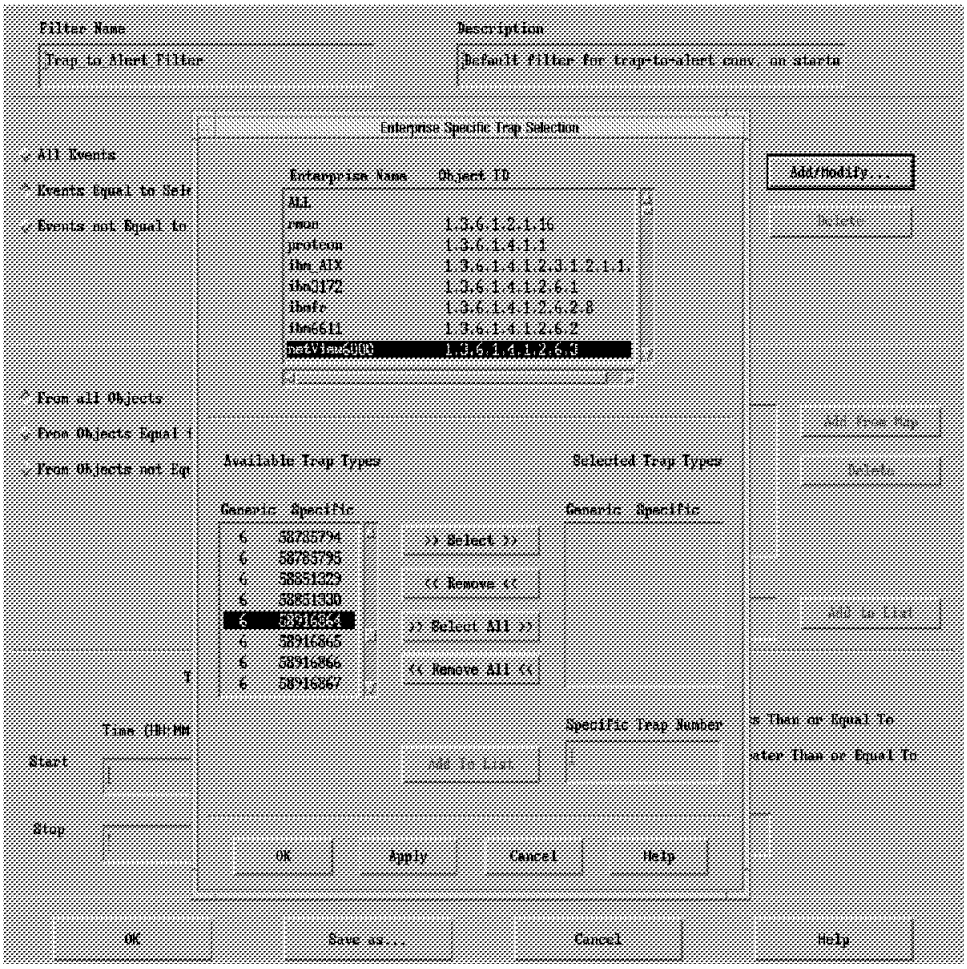


Figure 110. Filter Editor Including Browse of Generic/Specific

Select the activate button to start this filter.

Figure 111 on page 151 shows the default code point definitions for the Node Down event.

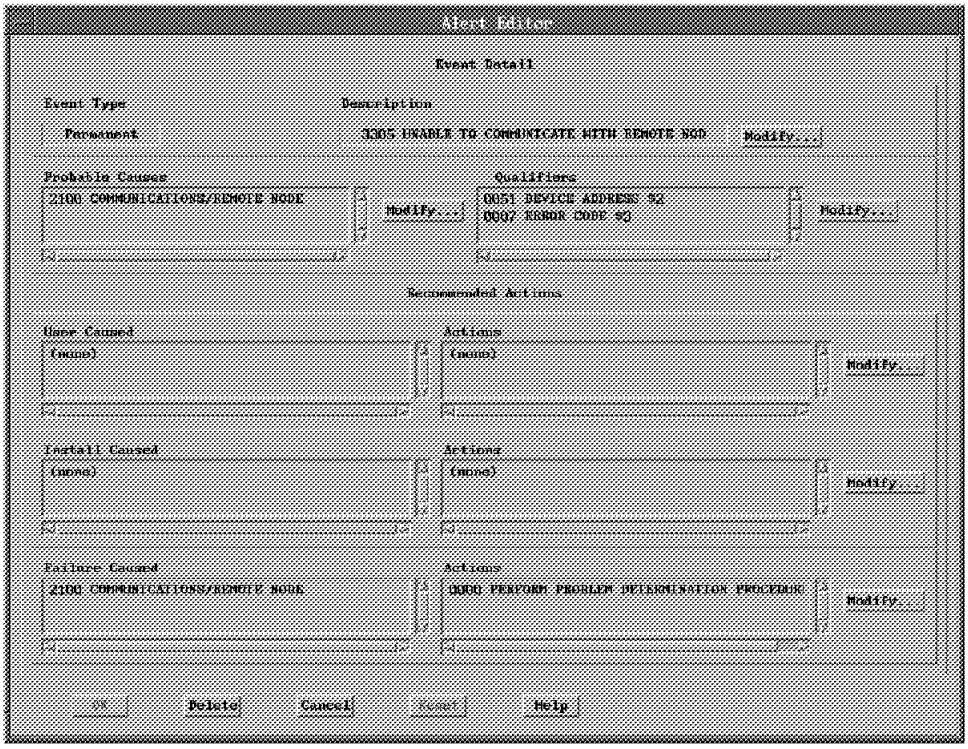


Figure 111. Code Points for NODE DOWN Event

To simulate a node down problem, type event -E 58916865, and you will see something similar to the panels displayed in Figure 112 and Figure 113 on page 152. As we are using default S/390 NetView code point entries, we have not had to make any changes to the S/390 NetView code point tables.

```
 N E T V I E W     SESSION DOMAIN: RABAN    WTWKSH8   04/04/94 14:57:15
 NPDA-45A      * RECOMMENDED ACTION FOR SELECTED EVENT *   PAGE  1 OF  1
  RABAN       RA6003CP     RS60003T     RS60003      RS60003
             +--------+   +--------+   +--------+   +--------+
  DOMAIN     |  SP    |---|  TP    |---|  DEV   |---|  DEV   |
             +--------+   +--------+   +--------+   +--------+

 USER    CAUSED - NONE

 INSTALL CAUSED - NONE

 FAILURE CAUSED - COMMUNICATIONS/REMOTE NODE
        ACTIONS - I000 - PERFORM PROBLEM DETERMINATION PROCEDURES

 ENTER ST (MOST RECENT STATISTICS), DM (DETAIL MENU), OR D (EVENT DETAIL)

  ???
CMD==>
```

Figure 112. S/390 NetView Node Down Recommended Action

```
N E T V I E W       SESSION DOMAIN: RABAN    WTWKSH8   04/04/94 14:58:26
NPDA-43S                  * EVENT DETAIL *              PAGE  1 OF  2

 RABAN       RA6003CP     RS60003T     RS60003      RS60003
            +--------+   +--------+   +--------+   +--------+
 DOMAIN     |  SP    |---|  TP    |---|  DEV   |---|  DEV   |
            +--------+   +--------+   +--------+   +--------+

SEL# TYPE AND NAME OF OTHER RESOURCES ASSOCIATED WITH THIS EVENT:
( 1) DEV     RS60002.ITSO.RAL.IBM.COM

DATE/TIME: RECORDED - 04/04 14:52    CREATED - 04/04/94 14:51:39

EVENT TYPE: PERMANENT

DESCRIPTION: UNABLE TO COMMUNICATE WITH REMOTE NODE

PROBABLE CAUSES:
   COMMUNICATIONS/REMOTE NODE

ENTER A (ACTION), SEL# (CORRELATED EVENTS), OR DM (DETAIL MENU)

???
CMD==>
```

Figure 113. S/390 NetView Node Down Alert Detail

4.14 S/390 NetView Code Point Customization

In order to have desired information on the S/390 NetView host, we have to make complementary changes to the S/390 NetView hardware monitor. To make the required changes, do the following from a S/390 NetView panel:

- Locate the BNJxxUTB member in the BNJPNL1 file.

  This can be done from S/390 NetView NCCF. The command and its result, are shown in **1** of Figure 114. The code point source in our sample is located in NETVIEW.NV24.BNJPNL1.

```
NCCF               N E T V I E W    RABAN WTWKSH8  04/04/94 11:30:24 A
          1
* RABAN    LISTALC BNJPNL1
' RABAN
CNM299I
DDNAME   DATA SET NAME                                          DISP
-------- ------------------------------------------------------ --------
BNJPNL1  NETVIEW.V2R4MO.BNJPNL1                                 SHR,KEEP
           2
* RABAN    CPTBL MEMBER=BNJ92TBL,TEST
- RABAN    CNM736I TEST OF CODE POINT MEMBER BNJ92TBL WAS SUCCESSFUL
          3
* RABAN    CPTBL MEMBER=BNJ92TBL
- RABAN    DSI633I CPTBL COMMAND SUCCESSFULLY COMPLETED
- RABAN    BNJ192I CODE POINT TEXT MAY HAVE BEEN CHANGED - CONSIDER RESTARTING
           YOUR NPDA SESSION IF YOU HAVE NOT DONE SO
--------------------------------------------------------------------------------
???
```

Figure 114. From S/390 NetView: Locating the S/390 NetView Code Point Tables

The structure of the member name format is BNJxxUTB where "xx" is the number of the subvector. Figure 115 on page 153, shows an update **1** to BNJ92UTB via S/390 TSO.

```
EDIT ---- NETVIEW.NV24.BNJPNL1(BNJ92UTB) - 01.04 ------- COLUMNS 001 072
COMMAND ===>                                              SCROLL ===> CSR
****** *************************** TOP OF DATA ************************
********************************************************************************
*                                                                              *
* TABLE NAME:  BNJ92UTB                                                        *
*                                                                              *
* DESCRIPTION:  THIS MEMBER IS USED TO GENERATE ALERT DESCRIPTION      *
*               CODE POINTS WHICH SUPPLEMENT THOSE SHIPPED BY IBM.     *
*               IT IS %INCLUDED BY BNJ92TBL.  REFER TO THE             *
*               NETVIEW CUSTOMIZATION GUIDE FOR THE FORMAT OF ENTRIES  *
*               AND MORE INFORMATION.                                  *
* (C) COPYRIGHT IBM CORP. 1988, 1994                                           *
*                                                                              *
********************************************************************************
*
*E000 N TEXT FOR E000 GOES HERE - MAX = 40 CHARS;
*E000 N ABOVE > 25 CHARS - ABBREV;
*E001 N TEXT SHORT - NO ABBREV;
1
E610 N ITSO E SNA NOW AVAILABLE;
**** ABBREVIATED ENTRY (SECOND ONE FOR THE SAME CODE POINT) IS REQUIRED
**** IF TEXT > 25 CHARS.  OTHERWISE DO NOT ADD ABBREVIATED ENTRY.
*************************** BOTTOM OF DATA ****************************
```

Note that this file is included by BNJxxTBL; that is why the command in the previous panel could point to BNJxxTBL.

- Using S/390 TSO, make and then save the changes to the BNJ92UTB.
- Using S/390 NetView, use the commands shown at **2** and **3** in Figure 114 on page 152, to test and then implement the changes.

Repeat the same operation for the code point tables for 81, 92, 93, 94, 95, and 96 that are mentioned in Table 12 on page 142.

When these changes are made, you will need to exit and then re-enter the hardware monitor display, and you will then see the changes to the alert displays, reflecting the user code points. The following figures show the results for the previously-used app_sendtrap example.

```
N E T V I E W          SESSION DOMAIN: RAPAN    WTWKSH6   11/17/94 18:47:18
NPDA-30A                      * ALERTS-DYNAMIC *

    DOMAIN RESNAME  TYPE TIME  ALERT DESCRIPTION: PROBABLE CAUSE
     RAPAN RA6003CP*DEV  18:47 SNMP RESOURCE PROBLEM: UNDETERMINED
     RAPAN RA6003CP*DEV  18:47 ITSC E SNA NOW AVAILABLE: ITSC P MEMORY
     RAPAN RA6003CP*DEV  18:47 SNMP RESOURCE PROBLEM: UNDETERMINED
     RAPAN RA6003CP*DEV  18:47 ITSC E SNA DOWN: ITSC P MEMORY2
     RAPAN RA6003CP*DEV  18:40 SNMP RESOURCE PROBLEM: UNDETERMINED
     RAPAN RA6003CP*DEV  18:40 SNMP RESOURCE PROBLEM: UNDETERMINED
     RAPAN RA6003CP*DEV  18:40 SNMP RESOURCE PROBLEM: UNDETERMINED
     RAPAN RA6003CP*DEV  18:40 SNMP RESOURCE PROBLEM: UNDETERMINED
     RAPAN RA6003CP*DEV  18:32 SNMP RESOURCE PROBLEM: UNDETERMINED
     RAPAN RA6003CP*DEV  18:32 SNMP RESOURCE PROBLEM: UNDETERMINED
     RAPAN RA6003CP*DEV  18:32 SNMP RESOURCE PROBLEM: UNDETERMINED
     RAPAN RA6003CP*DEV  18:32 SNMP RESOURCE PROBLEM: UNDETERMINED
     RAPAN RA6003CP*DEV  18:32 NO COMM WITH REMOTE NODE: COMM/REMOTE NODE
     RAPAN RA6003CP*DEV  18:32 NO COMM WITH REMOTE NODE: COMMUNICATIONS INTF
     RAPAN RA6003CP*DEV  18:30 PROBLEM RESOLVED: UNDETERMINED

 DEPRESS ENTER KEY TO VIEW ALERTS-STATIC

 ???
CMD==>
```

Figure 116. Alerts Dynamic Shows User-Alerts Arrived (ITSC)

```
N E T V I E W          SESSION DOMAIN: RAPAN    WTWKSH6   11/17/94 18:47:43
NPDA-45A           * RECOMMENDED ACTION FOR SELECTED EVENT *     PAGE  1 OF  1
 RAPAN       RA6003CP     RS60003T     RS60003      RS60003
            +--------+   +--------+   +--------+   +--------+
 DOMAIN     |  SP    |---|  TP    |---|  DEV   |---|  DEV   |
            +--------+   +--------+   +--------+   +--------+

USER    CAUSED - ITSC U APPLICATION DOWN
       ACTIONS - ITSC R RESTART APPLICATION

INSTALL CAUSED - ITSC I MEMORY USED
       ACTIONS - ITSC R RESTART APPLICATION

FAILURE CAUSED - ITSC F SOFTWARE DOWN
       ACTIONS - ITSC R RESTART APPLICATION




 ENTER ST (MOST RECENT STATISTICS), DM (DETAIL MENU), OR D (EVENT DETAIL)

 ???
CMD==>
```

Figure 117. Example of Recommended Action with User Code Points

```
N E T V I E W          SESSION DOMAIN: RAPAN    WTWKSH6   11/17/94 18:47:58
NPDA-43S                      * EVENT DETAIL *                 PAGE  1 OF  2

 RAPAN       RA6003CP     RS60003T     RS60003      RS60003
            +--------+   +--------+   +--------+   +--------+
 DOMAIN     |  SP    |---|  TP    |---|  DEV   |---|  DEV   |
            +--------+   +--------+   +--------+   +--------+

SEL# TYPE AND NAME OF OTHER RESOURCES ASSOCIATED WITH THIS EVENT:
( 1) DEV     RS60003.ITSO.RAL.IBM.COM

DATE/TIME: RECORDED - 11/17 18:47    CREATED - 11/17/94 18:45:49

EVENT TYPE: PERFORMANCE

DESCRIPTION: ITSC E SNA DOWN

PROBABLE CAUSES:
   ITSC P MEMORY2

ENTER A (ACTION), SEL# (CORRELATED EVENTS), OR DM (DETAIL MENU)

 ???
CMD==>
```

Figure 118. Example of SNA Down Event Detail with User Code Points - Page 1

```
N E T V I E W          SESSION DOMAIN: RAPAN    WTWKSH6   11/17/94 18:48:06
NPDA-43S                      * EVENT DETAIL *                 PAGE  2 OF  2

 RAPAN       RA6003CP     RS60003T     RS60003      RS60003
            +--------+   +--------+   +--------+   +--------+
 DOMAIN     |  SP    |---|  TP    |---|  DEV   |---|  DEV   |
            +--------+   +--------+   +--------+   +--------+

QUALIFIERS:
   1)  LOCATION ITSC Raleigh









UNIQUE ALERT IDENTIFIER: PRODUCT ID - 5696-7310  ALERT ID - C86245A8

ENTER A (ACTION) OR DM (DETAIL MENU)

 ???
CMD==>
```

Figure 119. Example of SNA Down Event Detail with User Code Points - Page 2

```
N E T V I E W          SESSION DOMAIN: RAPAN    WTWKSH6   11/17/94 18:48:23
NPDA-45A           * RECOMMENDED ACTION FOR SELECTED EVENT *     PAGE  1 OF  1
 RAPAN       RA6003CP     RS60003T     RS60003      RS60003
            +--------+   +--------+   +--------+   +--------+
 DOMAIN     |  SP    |---|  TP    |---|  DEV   |---|  DEV   |
            +--------+   +--------+   +--------+   +--------+

 USER    CAUSED - ITSC U APPLICATION UP
        ACTIONS - ITSC R MONITOR APPLICATION

 INSTALL CAUSED - ITSC I MEMORY FREE
        ACTIONS - ITSC R MONITOR APPLICATION

 FAILURE CAUSED - ITSC F SOFTWARE HAS RESTARTED
        ACTIONS - ITSC R MONITOR APPLICATION




 ENTER ST (MOST RECENT STATISTICS), DM (DETAIL MENU), OR D (EVENT DETAIL)

 ???
CMD==>
```

Figure 120. Example of SNA Up Event Detail with User Code Points - Page 1

```
N E T V I E W          SESSION DOMAIN: RAPAN    WTWKSH6   11/17/94 18:48:29
NPDA-43S                      * EVENT DETAIL *                 PAGE  1 OF  2

 RAPAN       RA6003CP     RS60003T     RS60003      RS60003
            +--------+   +--------+   +--------+   +--------+
 DOMAIN     |  SP    |---|  TP    |---|  DEV   |---|  DEV   |
            +--------+   +--------+   +--------+   +--------+

 SEL# TYPE AND NAME OF OTHER RESOURCES ASSOCIATED WITH THIS EVENT:
 ( 1) DEV     RS60003.ITSO.RAL.IBM.COM

 DATE/TIME: RECORDED - 11/17 18:47    CREATED - 11/17/94 18:45:58

 EVENT TYPE: PERMANENT

 DESCRIPTION: ITSC E SNA NOW AVAILABLE

 PROBABLE CAUSES:
    ITSC P MEMORY

 ENTER A (ACTION), SEL# (CORRELATED EVENTS), OR DM (DETAIL MENU)

 ???
CMD==>
```

Figure 121. Example of SNA Up Event Detail with User Code Points - Page 1

```
N E T V I E W          SESSION DOMAIN: RAPAN    WTWKSH6   11/17/94 18:48:35
NPDA-43S                      * EVENT DETAIL *                  PAGE  2 OF  2

 RAPAN       RA6003CP     RS60003T     RS60003      RS60003
            +--------+   +--------+   +--------+   +--------+
 DOMAIN     |  SP    |---|  TP    |---|  DEV   |---|  DEV   |
            +--------+   +--------+   +--------+   +--------+

 QUALIFIERS:
    1)  LOCATION ITSC Raleigh









 UNIQUE ALERT IDENTIFIER: PRODUCT ID - 5696-7310  ALERT ID - 8597C1CC

 ENTER A (ACTION) OR DM (DETAIL MENU)

 ???
CMD==>
```

Figure 122. Example of SNA Up Event Detail with User Code Points - Page 2

4.14.1 Sending Commands from S/390 NetView to NetView for AIX

The following discussion assumes the version of S/390 NetView installed supports the NETVASIS command in the operator command line. This command indicates to S/390 NetView that the operator or shell - entered commands should be left as is from a case-sensitive perspective.

Commands can be sent from S/390 NetView to NetView for AIX with the RUNCMD. Figure 123 on page 158 illustrates how this can be done. You need to consider the following:

1. The case-sensitive requirements of the command that you send to NetView for AIX.
2. The name of the catcher application that will execute the command.
3. The name of the ACF/VTAM PU or CP that hosts the catcher.

A is the VTAM CP name in our example, and **B** is the name of the service point application catcher.

In command example **1**, the command failed, as S/390 NetView folded it to uppercase before it was sent to NetView for AIX.

2 sent the command to NetView for AIX with ASIS in front of the ovstatus command. This still failed as S/390 NetView folded the command again. **C**

3 was as per **1** but was prefixed with the NETVASIS command. This still fails, as the service point application has folded the command to lowercase.

Finally 4 works because we have said to S/390 NetView and NetView for AIX, "send the command as I have typed it". The command that was entered is shown at D.

```
 NCCF               N E T V I E W    RABAN WTWKSH8  04/04/94 11:24:11 A
1           A     B
 * RABAN  RUNCMD SP=RA6003CP,APPL=RS60003S,/USR/OV/BIN/OVSTATUS NETMON
 -          Executing RUNCMD "/USR/OV/BIN/OVSTATUS NETMON" *
 -          bsh: /usr/ov/bin/ovstatus: not found*
2                                 C
 * RABAN  RUNCMD SP=RA6003CP,APPL=RS60003S,ASIS/USR/OV/BIN/OVSTATUS NETMON
 -          Executing RUNCMD "ASIS/USR/OV/BIN/OVSTATUS NETMON" *
 -          bsh: asis/usr/ov/bin/ovstatus: not found*
3
 * RABAN  runcmd sp=ra6003cp,appl=rs60003s,/usr/OV/bin/ovstatus netmon
 -          Executing RUNCMD "/usr/OV/bin/ovstatus netmon" *
 -          bsh: /usr/ov/bin/ovstatus: not found*
4
 * RABAN  runcmd sp=ra6003cp,appl=rs60003s,asis/usr/OV/bin/ovstatus netmon
 -          Executing RUNCMD "asis/usr/OV/bin/ovstatus netmon" *
 -           object manager name: netmon*
 -           behavior:            OVs_WELL_BEHAVED*
 -           state:               RUNNING*
 -           PID:                 21890*
 -           last message:        Initialization complete.*
 -           exit status:         -*
 -          *
 ------------------------------------------------------------------------
 ???
D
netvasis runcmd sp=ra6003cp,appl=rs60003s,asis/usr/OV/bin/ovstatus netmon
```

Figure 123. RUNCMDs to NetView for AIX Service Point

4.15 AIX Error Log Interaction with NetView for AIX

The AIX error log is used by applications and the operating system to record status and problem information. This log is called /usr/adm/ras/errorlog, and it is controlled by the errdemon daemon.

It is useful to generate events using the information in the AIX error log.

We can use the trapgend daemon to make this information available to NetView for AIX.

4.16 trapgend Daemon

Trapgend is a SMUX subagent provided by NetView for AIX. It can be installed on local and remote machines via the NetView for AIX GUI.

Trapgend will convert AIX-alertable errors, into SNMP traps.

Note!

Do not confuse an AIX-alertable message with a NetView for AIX event-to-alert conversion. The NetView for AIX events originate as a result of traps; the AIX-alertable message originates as an entry in the AIX V3 errlog. A specific error log entry will be alertable if the template that defines it has Alert=True defined. Any application may write into the AIX V3 errlog.

The link between trapgend and errdemon, is made with the trap-notify command. This command is invoked for any AIX error that is logged.

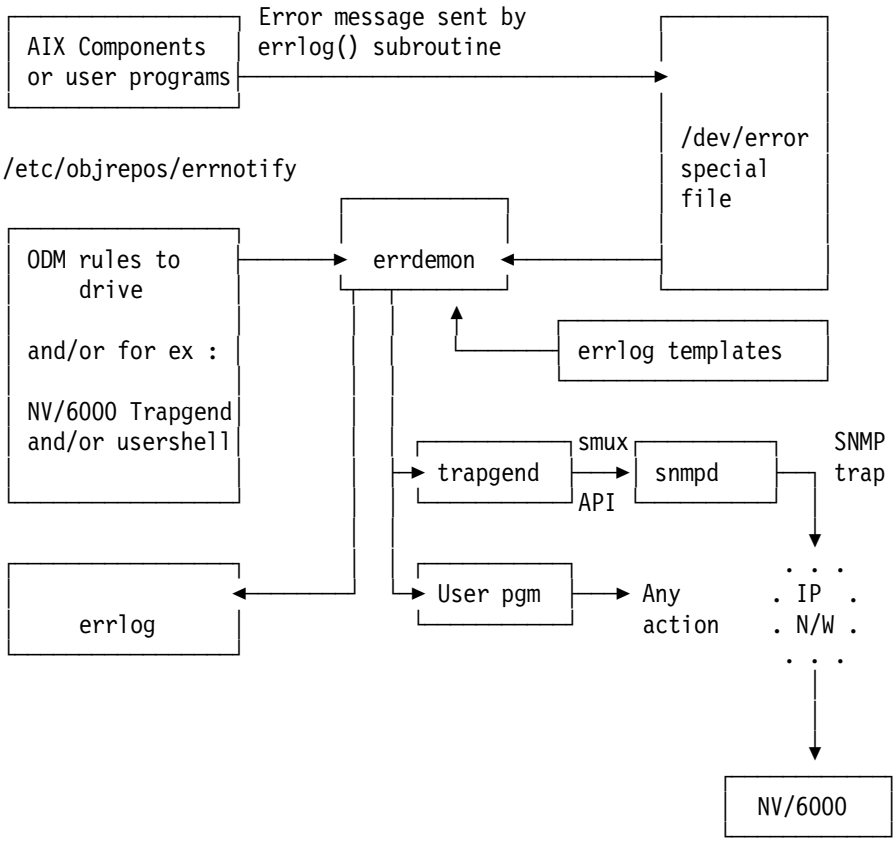


Figure 124. AIX Error Descriptions Passed to NetView for AIX. Whenever there is a message written to the AIX error log, if the template that defines it is flagged as alertable then the AIX message is forwarded to trapgend by the trap-notify process. Trapgend then converts the error into an SNMP trap which is broadcast on the network by snmpd.

4.16.1 AIX Error Log Examples

The following examples show how an application error message sent to the AIX error log is converted into a NetView for AIX event. The example will simulate an error being raised by an application running a batch job, where an error is raised during execution, but the job continues. Because the AIX error log can also be used to record actions, not just errors, it could be used as an audit trail facility.

Before we discuss them in detail, here are some useful commands for handling this environment.

Table 13. Useful AIX Error Log Commands	
Command	**Function**
errpt	Displays a summary of error log messages
errpt -a	Displays detailed list of messages
errpt -t	Displays all defined error message templates
errclear 0	Clears all entries in the error log
errpt -tF alert=1	Displays all alertable errors

4.16.2 Example of Using the AIX errlog to Generate Events

All messages written to the AIX error log have to go via the errlog() C function. In the following sections, we show an example of a program that allows this service to be called from the command line.

A template has to be created to hold the information for each error that is required to be entered in the AIX error log. The purpose of the template is to serve as criteria to access the data in the error log.

4.16.3 Converting Existing AIX Errors into Events

There are a number of predefined errors already configured as alertable. NetView for AIX also has a number of predefined events that trapgend will generate.

To list the errors set as alertable, type `errpt -tF alert=1`. All these errors will be converted into events via trapgend. We will follow the procedure for linking the AIX error to the event display window.

In this example when the errdemon is turned off, an entry is made in the errlog. If the error logging is being used to generate events then we need to monitor its activity. The first stage is to make this error alertable. To do this:

- To find the ID for the error, type `errpt -t | grep "Error logging turned off"`. This will show:

  `192AC071 ERRLOG_OFF TEMP 0 Error logging turned off"`

- To make this error alertable, type: `echo "= 192AC071:\n Alert=true" | errupdate"`

  To list the errors set as alertable type `errpt -tF alert=1` and see if the error is set as alertable.

  To see the event defined for this error, from the NetView for AIX pull-down menus :

  - Select **Options-> Event Configuration-> Trap Customization**.
  - Select Enterprise Name **netView6000subagent**.
  - After toggling the Hex Display button, search for the specific number: 192ac071.
  - Enter the rest of the information shown in Figure 125 on page 161.

To stop the error daemon enter `/usr/lib/errstop`.

To restart the error daemon enter: `/usr/lib/errdemon`. This will generate the event shown in Figure 126 on page 161.

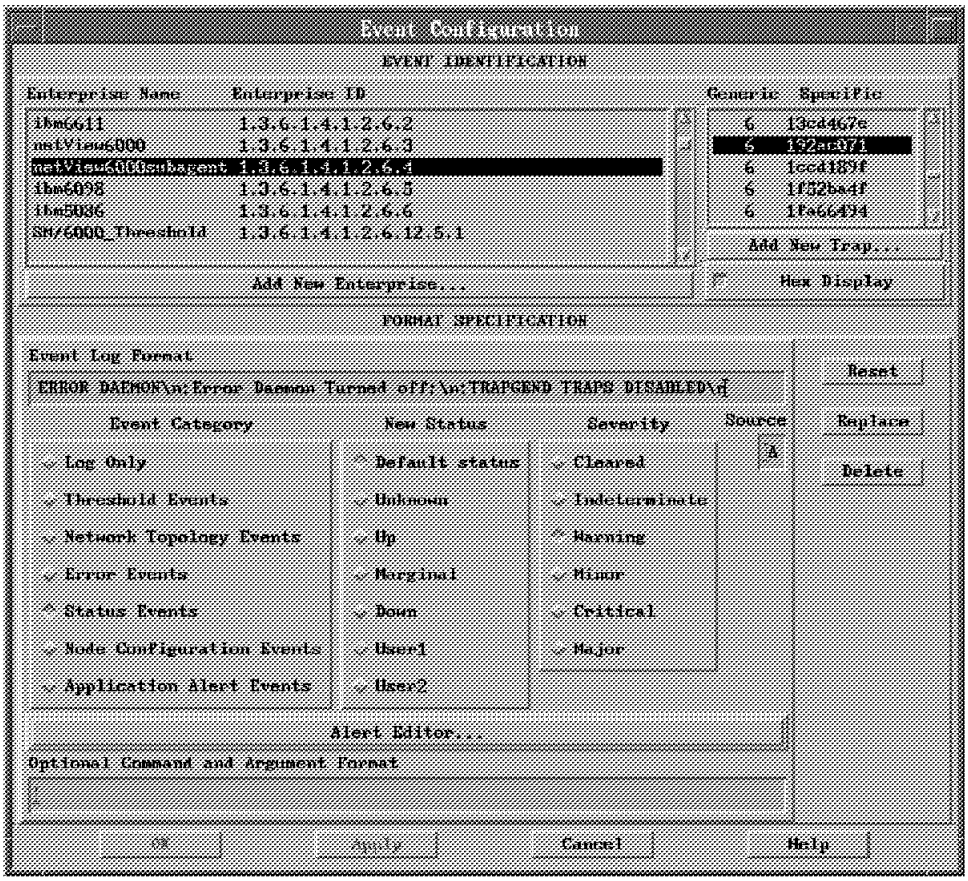


Figure 125. Configuration for trapgend Event

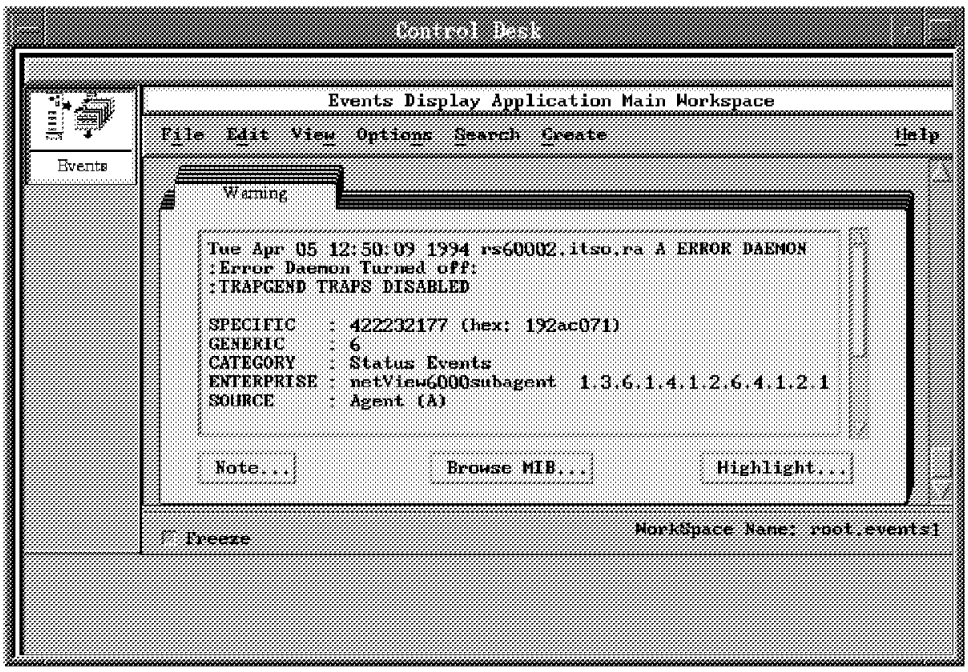


Figure 126. Daemon Down Event for errlogger

4.16.4 Example of Creating New Error Definitions

Earlier, in 4.13.3, “Customizing NetView for AIX Aimed at S/390 Host Alerts” on page 142, an example was shown of creating a relationship which enabled useful code points by NetView for AIX and S/390 NetView. The following approach is similar and presented here as a convenience to readers who may have moved directly to this chapter.

The following example shows how to create a new error and define the relevant event information.

The procedure we followed was as follows:

- Create a file called errinstall.sample.

errinstall.sample

```
*
*  SAMPLE CONFIGURATION FILE:  errinstall.sample
*
*  SET  D  to add Detailed Data      message
*       E  to add Error Description  message
*       F  to add Failure Cause      message
*       I  to add Install Cause      message
*       P  to add Probable Cause     message
*       R  to add Recommended Action message
*       U  to add User Cause         message
*
*  The message id is 4 characters and treated as hex digits
*  The message text should not exceed 40 characters
*  You can have multiple different messages within each SET
*

SET D
E620 ”ITSO Application Error”

SET E
E620 ”ITSO Batch Update Failed”

SET F
E620 ”ITSO Software Failure”

SET I
E620 ”ITSO Install Cause”

SET P
E620 ”ITSO Batch Application error”

SET R
E620 ”ITSO Check Batch Log file”

SET U
E620 ”ITSO User Cause”
```

- To place these code points into the AIX error message catalog, issue the command:

  ```
  errinstall -f errinstall.sample
  ```

- To verify the above, type errmsg -w ALL | grep E620. This will type out the entries.

- Create a file called errupdate.sample:

errupdate.sample

```
+ ITS_BATCH:
Class=          S
Log=            True
Report=         True
Alert=          false
Err_Type=       TEMP
Err_Desc=       E620
Prob_Causes=    E620
User_Causes=    E620
User_Actions=   E620
Inst_Causes=    E620
Inst_Actions=   E620
Fail_Causes=    E620
Fail_Actions=   E620
Detail_Data=    70, E620, ALPHA
Detail_Data=    70, E620, DEC
Detail_Data=    70, E620, HEX
```

- Now run the command errupdate errupdate.sample. You will get a message that says 1 entry has been added.

  AIX has generated a 32-bit ID during this process. We need to find out what it is to be able to use the template. The file has been placed in the / directory with an appended name of undo.

- Type cat errupdate.sample.undo to find the error ID.

errupdate.sample.undo

```
- 1f52ba4f:
```

  To check that the template entry was created, use the

  errpt -t | grep 1F52BA4F

  command. Note that the search argument for the GREP uses uppercase letters.

errpt -t | grep 1F52BA4F

```
1F52BA4F ITSO_BATCH TEMP    S    ITSO Batch update failed
```

  We now need to make this message alertable, so that it can be sent to trapgend via the trap-notify process.

- Type echo "= 1F52BA4F:/n" alert="true" | errupdate

  This makes the message alertable.

- To verify that it has succeeded, type: errpt -tF alert=1 | grep 1F52BA4F

  We now need to generate the error, and log it to the AIX error log. To do this we use the sample program in Figure 127 on page 164. Compile the program and then type:

  add_error 1F52BA4F "Test error"

- To verify that this has been sent to the error log, type:

  errpt | grep 1F52BA4F

You should also see an event with no known format in the NetView for AIX event display.

```
/*********************************************************************/
/**  AUTHOR     : Karl Yao, IBM Hong Kong                            **/
/**  Paul Fearn, IBM UK                                             **/
/**  PROGRAM    : add_errlog                                        **/
/**  DESCRIPTION: This C program logs an 8 characters hex number, and **/
/**               a 14 characters description to AIX error log      **/
/**  USAGE      : add_errlog error-id resource-name-description     **/
/**  NOTE     : To compile, use cc -O add_errlog.c -o add_errlog -lrts**/
/*********************************************************************/

#include <sys/errids.h>

unsigned ctohex(c)
                        char c;
{
if (c <= '9')  c = c - '0';
else if (c <= 'F')  c = (c - 'A') + 10;
else c = (c - 'a') + 10;
return(c);
}

main(argc, argv)
                        int argc;
                        char *argv[];
{
ERR_REC(80)             errbuf;
char                     *hex, *name;
unsigned                sum=0;
int                      i, shift=0, rc;

if (argc == 1) {
  printf("Usage: errlog  <8-digit hex number> <optional resource name>\n");
  return;
  }

hex = argv[1];
i = strlen(hex) - 1;
do {
  sum = (ctohex(hex[i]) << shift) + sum;
  shift = shift + 4;
  i = i - 1;
  }
while (i >= 0);

errbuf.error_id = sum;
name = argv[2];
if (strlen(name) == 0) name = "ERRLOG TEST";

strcpy( errbuf.resource_name, name );

rc = errlog( &errbuf, sizeof(errbuf) );
if (rc != 0)
                        printf("Errlog failed, rc=%d\n", rc);
else
                        printf("Errlog successful, rc=%d\n", rc);
}
```

Figure 127. add_errlog Sample Program to Write to the AIX Error Log

4.16.5 NetView for AIX Event Configuration

We now have to customize the event that is sent to NetView for AIX to provide some meaningful information. But to do this, we have to be able to identify the enterprise-specific event.

We have to convert the hex number into its decimal equivalent. Type the following on an AIX command line:

```
bc
obase=10
ibase=16
1F52BA4F
525515343
quit
```

The value 525515343 will be used to define the specific event for the error.

If the result is greater than x'80000000' then the number must be a negative. To calculate this value the event ID will be the negative decimal of x'100000000' minus the hex error ID.

For example, if the hex error ID is F2D2CB90 this value is greater than x'80000000'. To calculate the event ID, on the command line type the following:

```
bc
obase = 10
ibase = 16
100000000-F2D2CB90
221066352
quit
```

The correct value for the specific event number will be the negative of the final value, (for example, -221066352).

The event can now be added and customized in NetView for AIX.

To do this we do the following from the NetView for AIX pull-downs.

- Select **Options-> Event Configuration-> Trap Customization**.
- Select the **netView6000subagent** enterprise name.
- Select **Add New Trap**.
- Select **enterpriseSpecific**.
- Enter 525515343.
- Set Event Category to "Status Events".
- Set Severity to "Minor".
- Enter the following line into the "Event Log Format" field

  ```
  ERROR:\nDetailed Data: $%28\n$%14\n (errpt -al "Log Number")\n
  ```
- Select **Replace**.
- Select **Apply**.

Once the event has been configured, it can be referenced by clicking on **Hex Display**. The specific event can be found using the Error ID number (1F52BA4F).

To send the event, type the following:

```
add_errlog 1F52BA4F "Batch Module"
```

The event will appear similar to Figure 128 on page 166.

Once the event has been configured, the event can be referenced by clicking on **Hex Display**. The specific event can be found using the Error ID number (1F52BA4F).

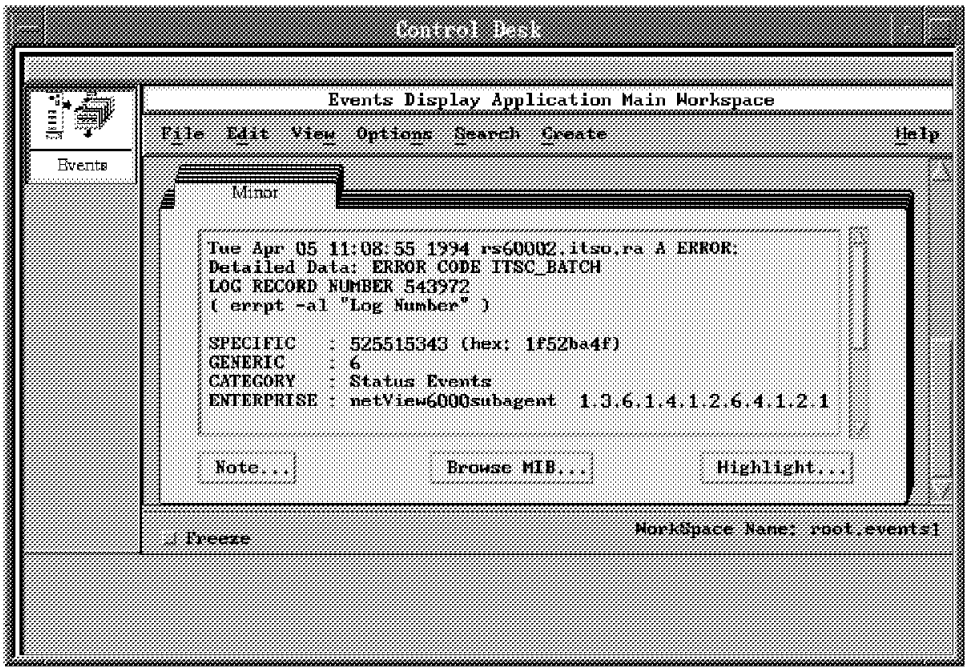


Figure 128. Configuration for trapgend Event

To view the error log entry for more details, record the LOG RECORD NUMBER from the event display window. In the example used, LOG NUMBER is 543972, type:

```
errpt -al 543972
```

This will display all details from the AIX errlog for the generated error as shown in the following figure.

```
---------------------------------------------------------------------------
ERROR LABEL:      ITSO_BATCH
ERROR ID:         1F52BA4F

Date/Time:        Tue Apr  5 10:28:13
Sequence Number: 543973
Machine Id:       000105681000
Node Id:          rs60002
Class:            S
Type:             TEMP
Resource Name:    Batch Module

Error Description
ITSO Batch Update Failed

Probable Causes
ITSO Batch Application Error

User Causes
ITSO User Cause

                  Recommended Actions
                  ITSO Check Batch Log File

Install Causes
ITSO Install Cause

                  Recommended Actions
                  ITSO Check Batch Log File

Failure Causes
ITSO Software Failure

                  Recommended Actions
                  ITSO Check Batch Log File

Detail Data
ITSO Application Error
```

Figure 129. AIX Error Log Display

Chapter 5. NetView for AIX Open Topology

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 can 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.

5.1 Open Topology Components

Figure 130 on page 171 shows the elements that make up the IP Topology and Open Topology parts of NetView for AIX. Notice the symmetry between the two halves of the diagram; for each function that is specifically to do with IP Topology, there is an equivalent non-IP function. We will examine the roles played by each of these component pairs:

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, 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, 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 described above.

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 below (5.4, “Network Discovery with Open Topology” on page 175).

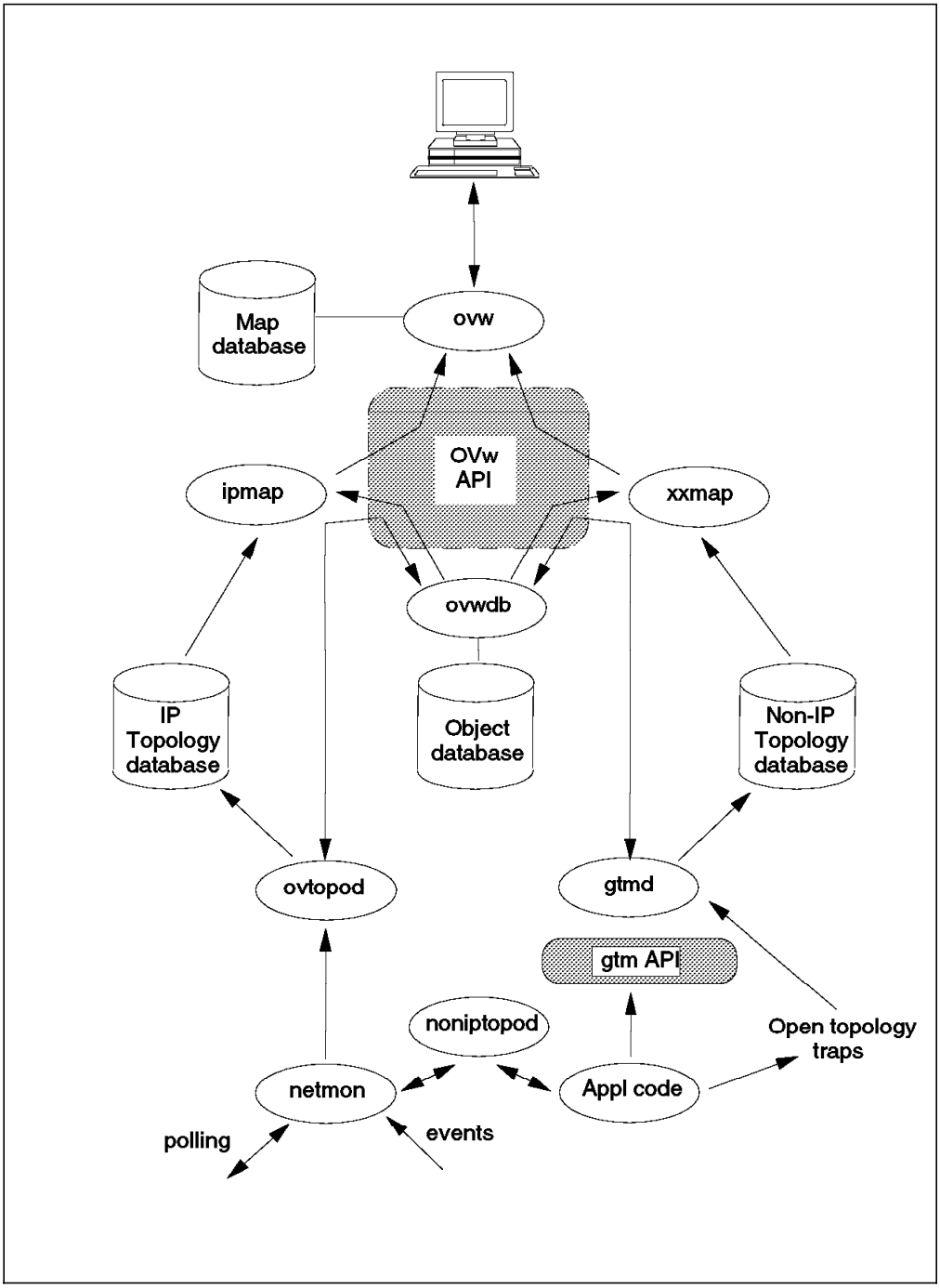


Figure 130. Components of IP and Open Topology

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 all defined in the Open Topology MIB. If you want to understand the elements of the MIB in more detail you should refer to Appendix A, “Open Topology MIB Reference” on page 241.

Open Topology support has been further enhanced in NetView for AIX by the addition of an API which may be used in place of the trap interface. We discuss this further below (5.6, “The Open Topology API” on page 179).

5.2 Applications Using Open Topology

Currently this technology is used by:

- LAN Network Manager/6000 (LNM/6000)

  LNM/6000 communicates with LNM for OS/2 to provide Token Ring configuration and performance management.

- LAN Management Utilities/6000 (LMU/6000)

  LMU/6000 communicates with LMU for OS/2 to provide control and management of OS/2 LAN Server, Novell Server and their client workstations.

- SNA Manager/6000

  SNA Manager communicates with NetView/390 to provide topology display and control of an SNA subarea network.

5.3 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 which 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 a 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 described above. In the case of FDDI, therefore we would define vertices using a protocol ID of 15.

Below we list the more common types of object defined by the open topology model. All of 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 Where 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 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 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 just 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. As an 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 5.5, “Open Topology Service Access Points” on page 176.

Figure 131 on page 174 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.

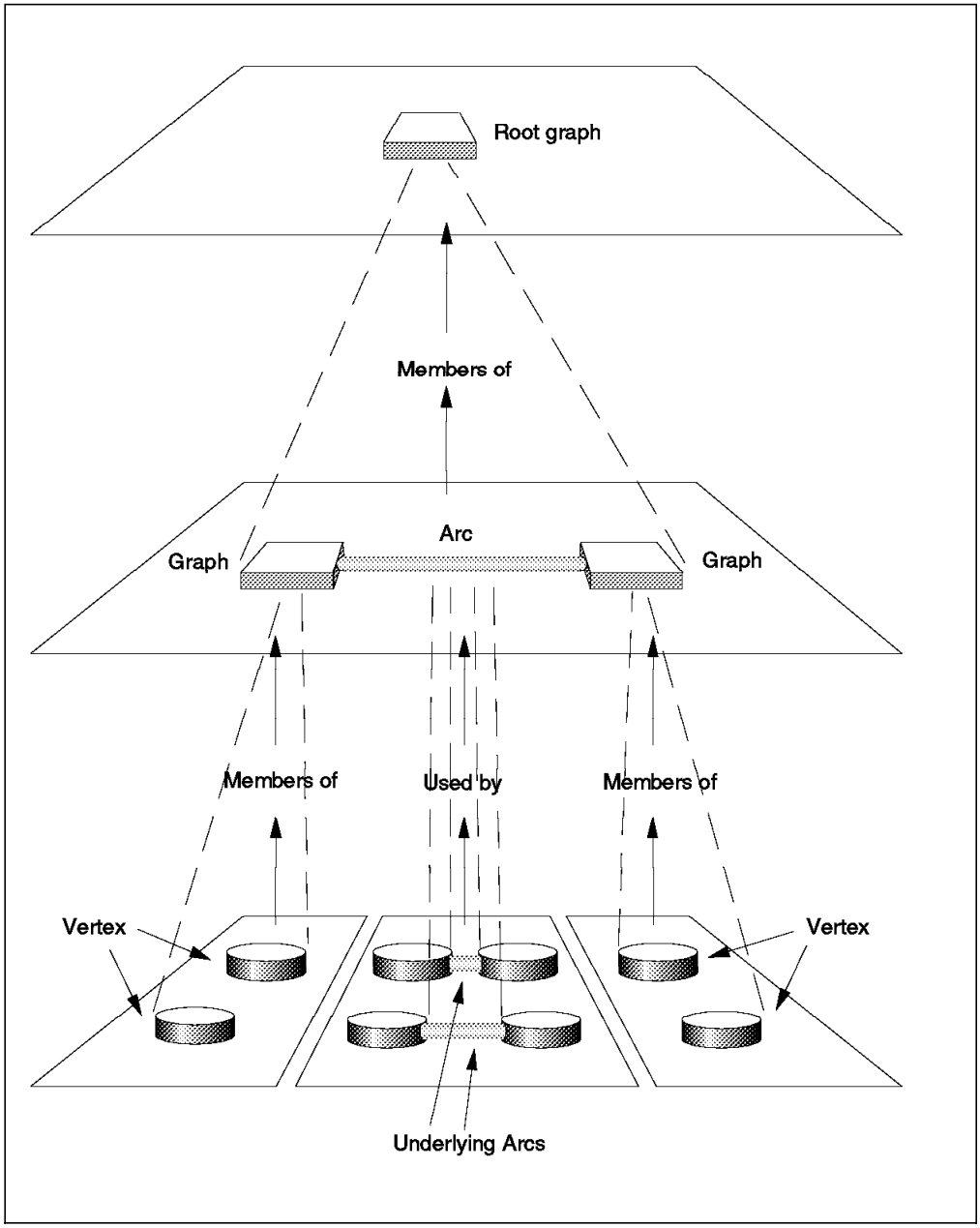


Figure 131. Some Elements of the Open Topology Model

5.3.1 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.

File `/usr/OV/conf/C/oid_to_sym` provides the mapping between object instances and icons.

5.4 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 132 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 pmd daemon.
2. noniptopod uses event registration to allow itself to be sent copies of all these discovery event. 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, from NetView for AIX, using the gtmd API information to the user.

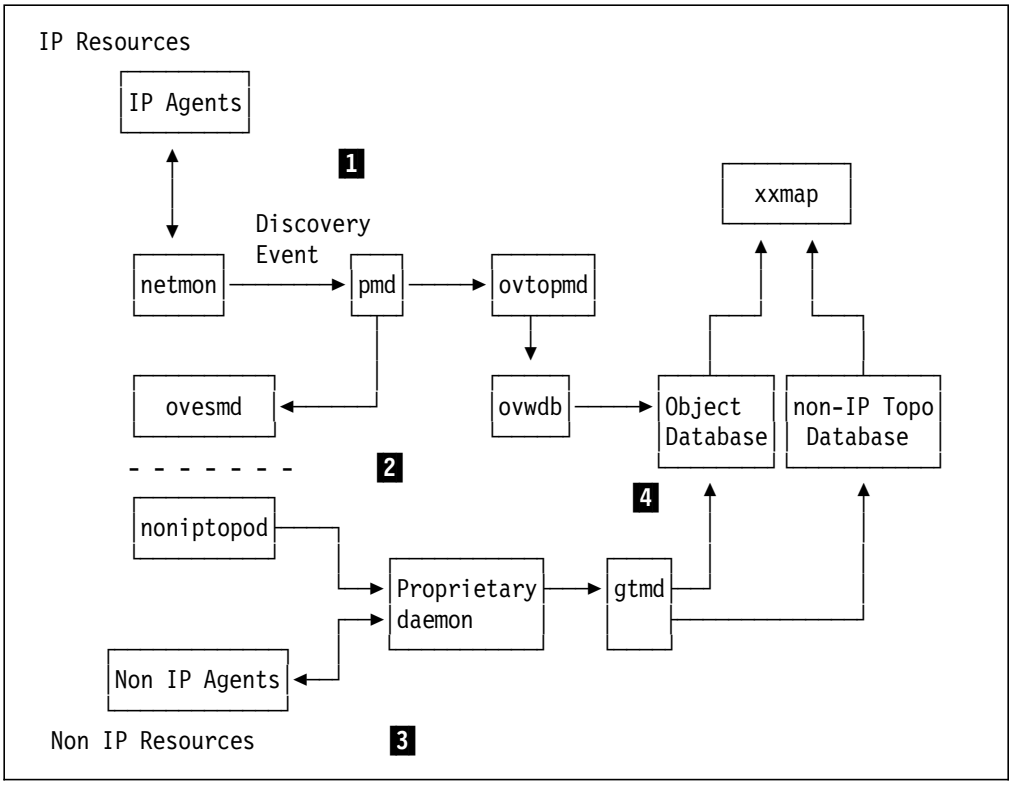


Figure 132. The Discovery Process and the Open Topology MIB

5.5 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 element(s) 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 an token-ring adapter card, that is *both*:

- Physically attached to a token-ring CAU, and,
- 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.

5.5.1 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 2 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 a 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 133 on page 177.

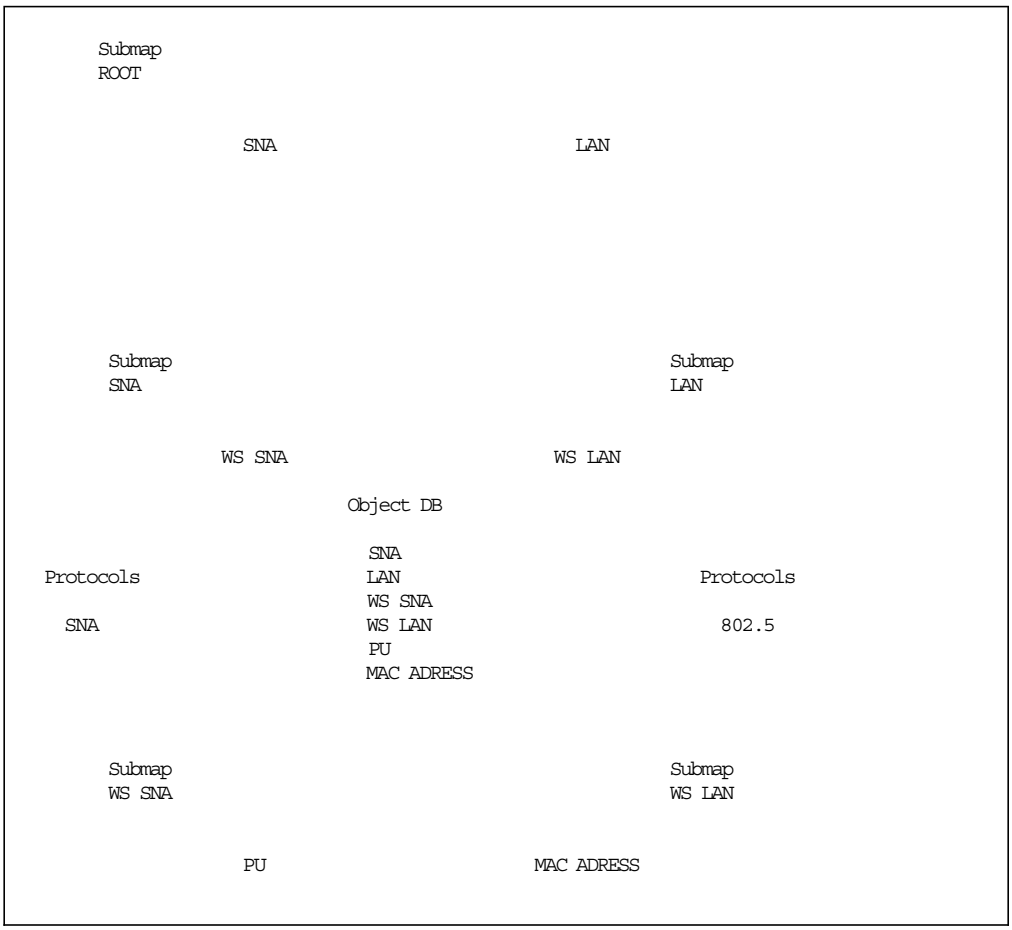


Figure 133. SNA and Physical Network Topologies with No Correlation

5.5.2 Open Topology Invocations

The above 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 a SNA object in the object database. xxmap adds a SNA submap symbol to the root map, associated with the SNA object in the object database. xxmap also creates a 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 it we have not yet told it which graph the WS_SNA graph is a member of, and 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.

5.5.3 Using Open Topology Correlation

Now that we have placed the SNA PU on to 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 is not available to either environment, about the existence of the other. 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 affect on the NetView for AIX submaps and object data is shown in Figure 134 on page 179.

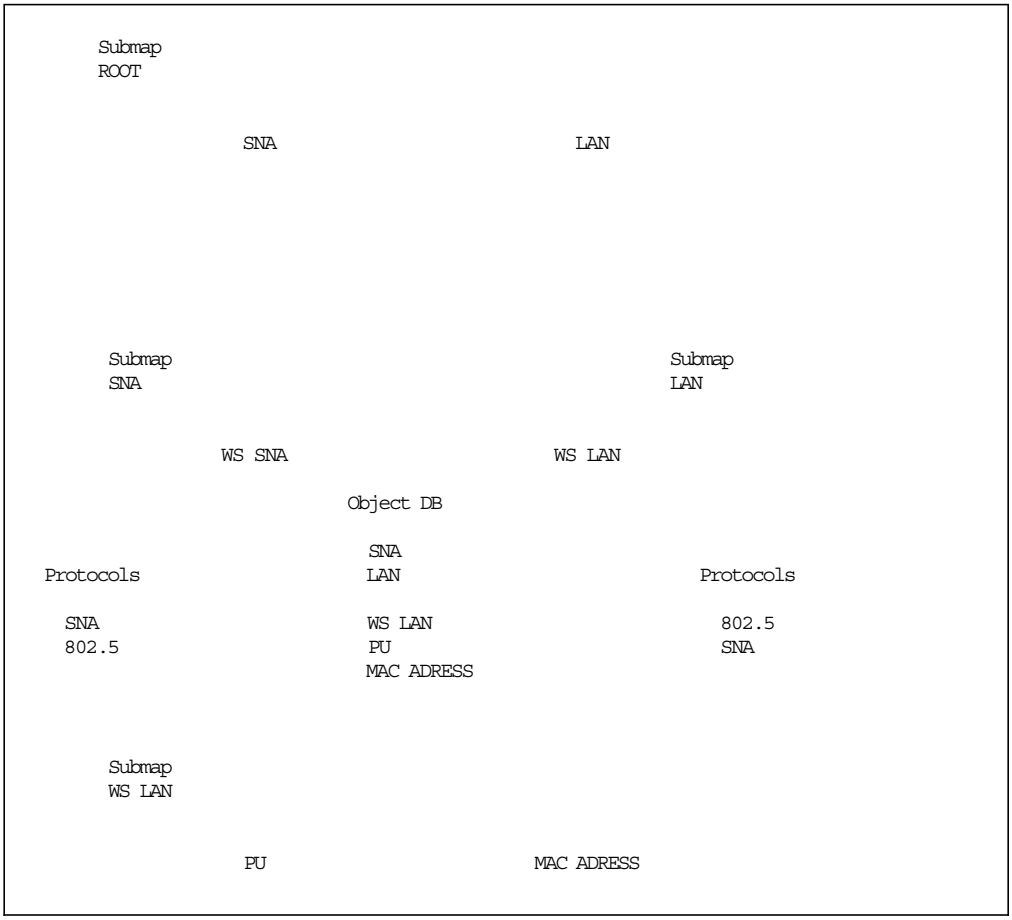


Figure 134. LNM/6000 and SNA/6000 with Correlation

The xxmap EUI application allows you to transfer between the SNA and the LAN submaps. This can be done by listing which protocols are being used by the workstation and switching to the one that you want.

5.6 The Open Topology API

With NetView for AIX V2.1 we are 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 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.

5.6.1 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 described above. 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.

5.7 Open Topology Samples

During this project, we developed two sample programs for driving the NetView for AIX open topology processes. Both programs can be executed from the command line, or by taking input from a file.

wtgtm This is a shell script which uses the SNMP Trap interface. It sends traps using the `snmptrap` command of NetView for AIX. This program is compatible with NetView for AIX V2 as well as NetView for AIX.

wtotapi1 This is a C program that uses the NetView for AIX open topology API.

Both programs are listed in Appendix C, “Open Topology Program Samples” on page 255.

The objectives of both programs is to provide a simple command-line technique to define and control network topology views, using the Open Topology support of NetView for AIX. Neither program exploits the full capability of the function, but they do 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 working example.

5.7.1 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.. The elements we want to map on the NetView for AIX display are therefore:

- 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.

5.7.1.1 Defining the Protocol ID

The first thing to consider is the protocol ID that we will use. As described above (5.3, “Terms and Concepts” on page 172) the protocol IDs are defined in 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"
```

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".

5.7.1.2 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:

```
1.3.6.1.4.1.2.8.2.1:Software:Process
1.3.6.1.4.1.2.8.2.2::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.

5.7.1.3 Defining the Topology

Although the sample programs that we developed (wtgtm and wtotapi1 - see 5.7, “Open Topology Samples” on page 180) can be invoked from the command line, it is easier to place all the commands in one file to be invoked together. Also, to save confusion, we will only use the "wtotapi1" version in this example.

The command file used to create our NFS sample is as shown below:

```
prefix 1.3.6.1.4.1.2.8.1.                                    1
prot 1                                                       2
add graph NFS Network:Star rowcol                            3
focus NFS                                                    4
add graph rs60003.nfs Software:Process tree                  5
add graph rs60005.nfs Software:Process tree
focus rs60003.nfs
add box /usr/local "Device:Hard Disk" rowcol                 6
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                                 7
add arc   rs60002 /usr/local
add arc   rs60004 /usr/local
add arc   rs60002 /wtprint
focus rs60005.nfs                                            8
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 135. Command File nfsmap Using wtotapi1 Sample Code

wtotapi1 reads commands from standard input, so we invoke the above command file by entering: `wtotapi1 < nfsmap`.

Some notes on the contents of the nfsmap command file:

1. We want to use our own protocol OID (see 5.7.1.1, “Defining the Protocol ID” on page 181) 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 above sequence of instructions is shown below:

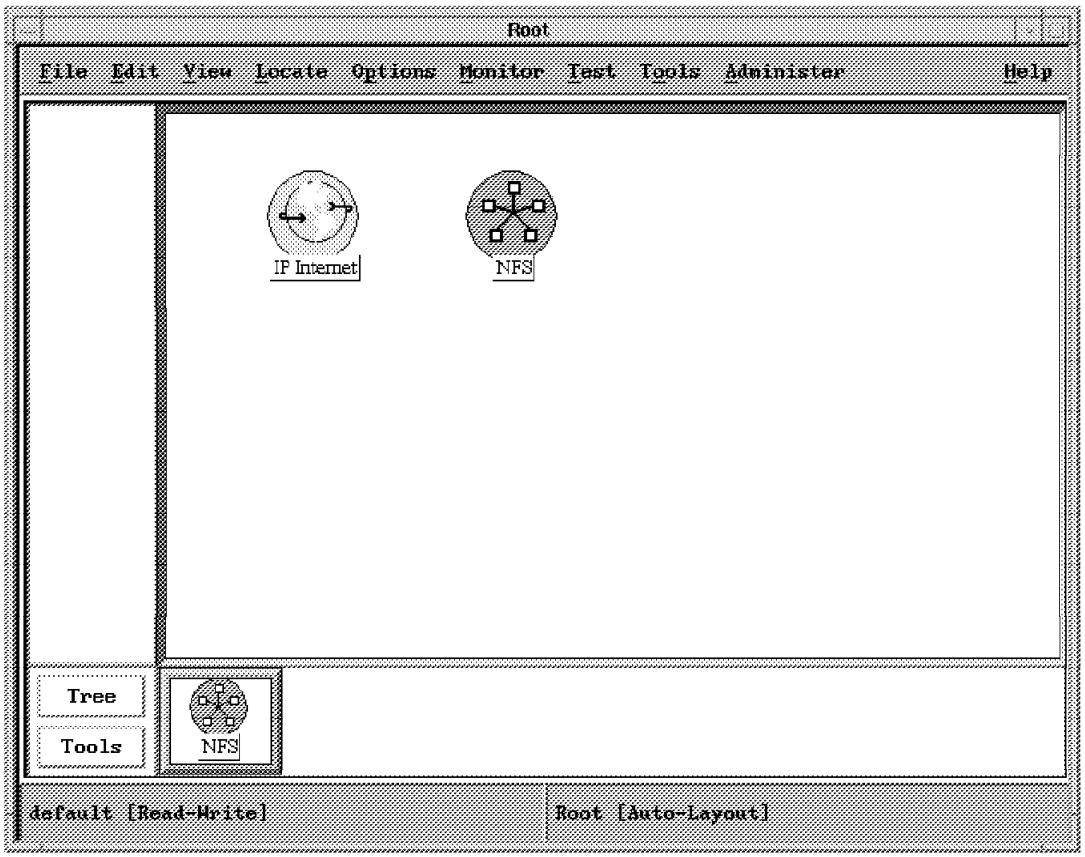


Figure 136. The Root Submap as Updated by this Example

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:

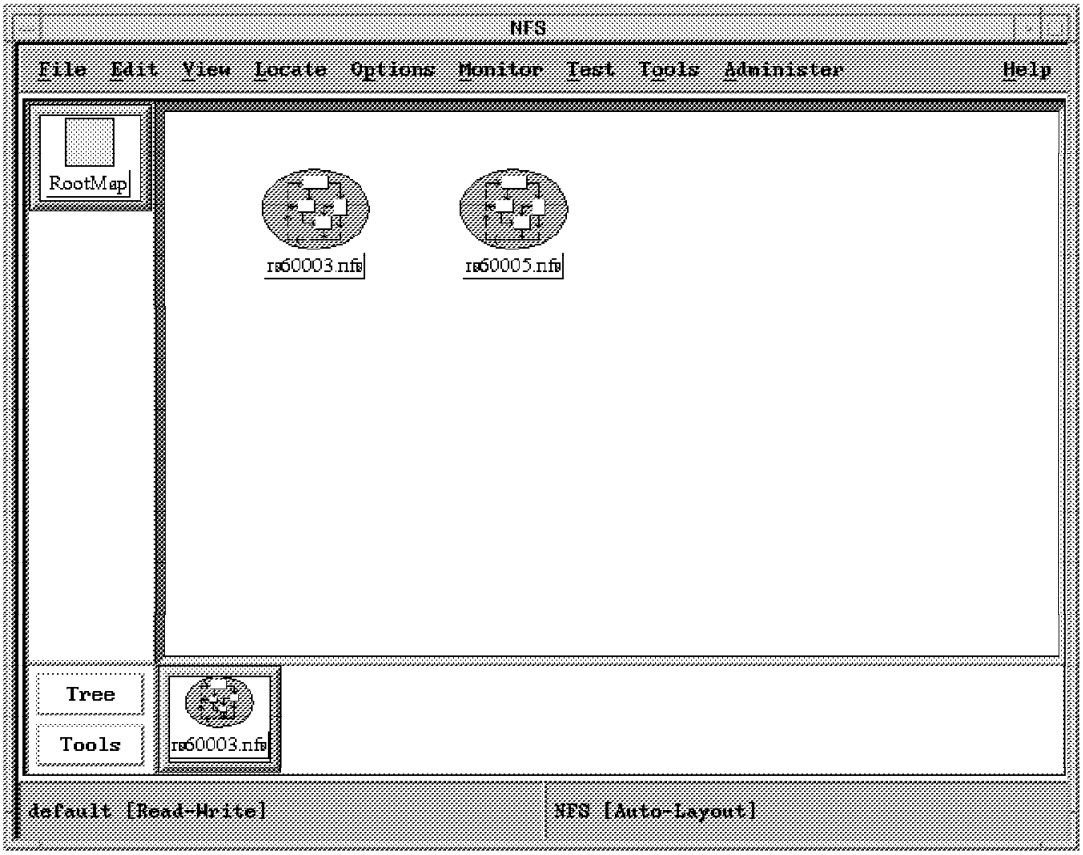


Figure 137. The NFS Server Submap

Our two servers are represented by Software:Process symbols. We explode rs60003.nfs:

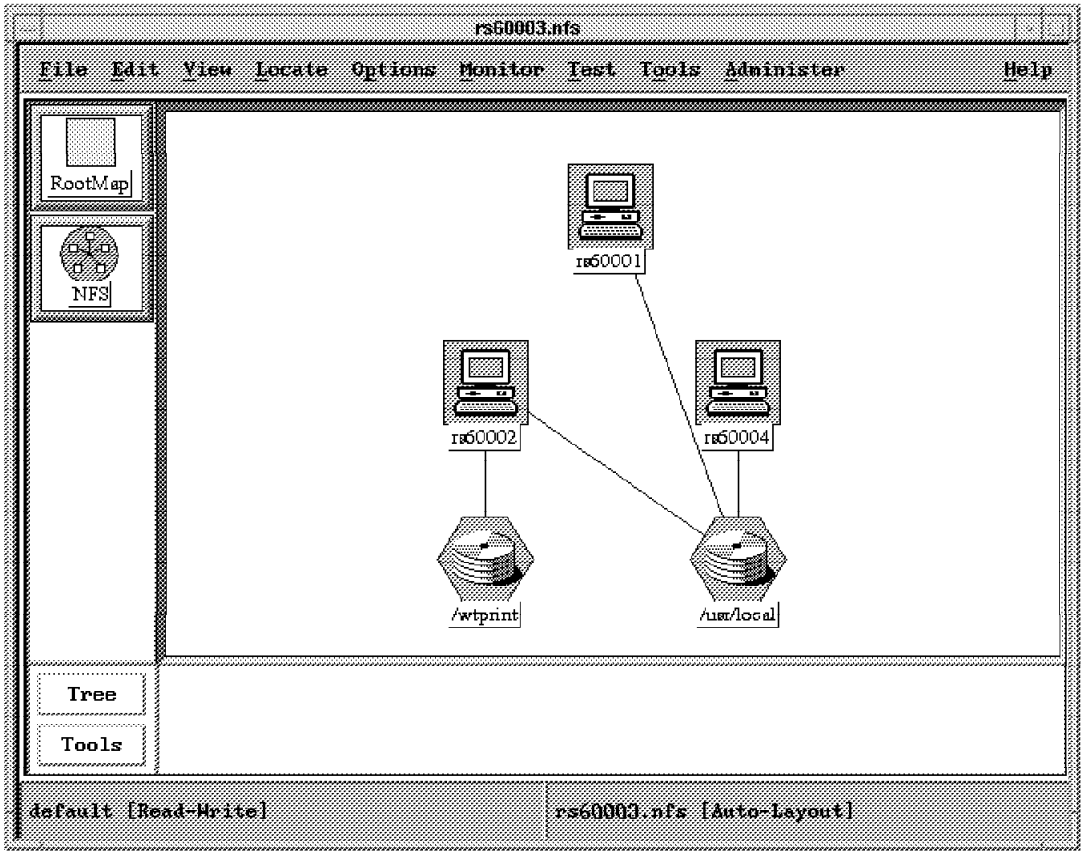


Figure 138. The Mounted File System Connections

Here we see symbols representing the exported file systems and the NFS client systems that have mounted them.

5.7.1.4 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 up. 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 to wtotapi1:

```
prefix 1.2.3.4.
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 139. 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 in Figure 138 on page 185). For example if we explode the rs60002 symbol, we will see the screen in Figure 140 on page 187.

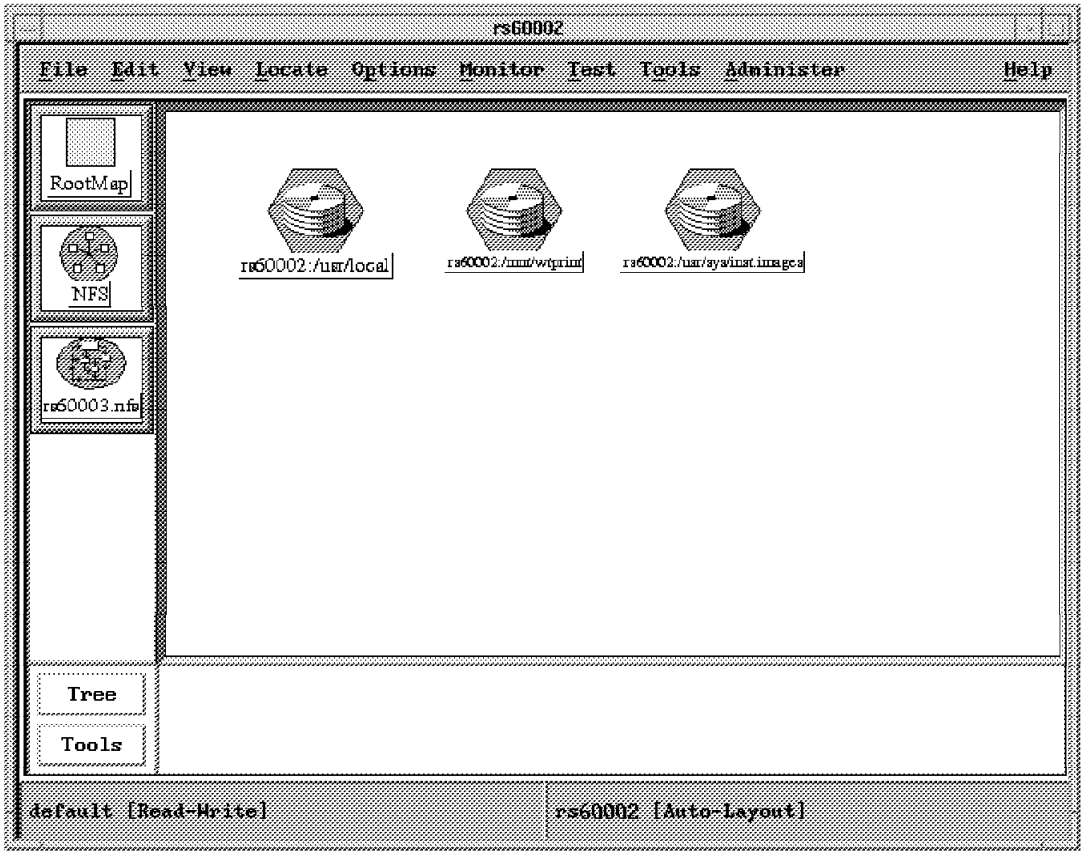


Figure 140. rs60002 Submap, Showing Vertex Symbols

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.

5.7.1.5 Setting Vertex Status

So far all of our work has been happening on the manager (NetView for AIX). We have created a picture of the NFS network, but as yet it is just 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 ″smart 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 four 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 wtgtm or 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. Place a copy of wtgtm on the agent system and use it to remotely update gtmd on the manager.
4. Execute the `snmptrap` command to set the status directly.

All four 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 (wtgtm, and option 4) is the vehicle. We elected to go for option 4. The advantage is that because the `snmptrap` command is part of Systems Monitor, it will work on systems to which wtgtm or 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 below (Figure 141 on page 189). This entry causes the Systems Monitor agent (the `sysmond` daemon) 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 pressing the Threshold Actions button. Figure 142 on page 190 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).

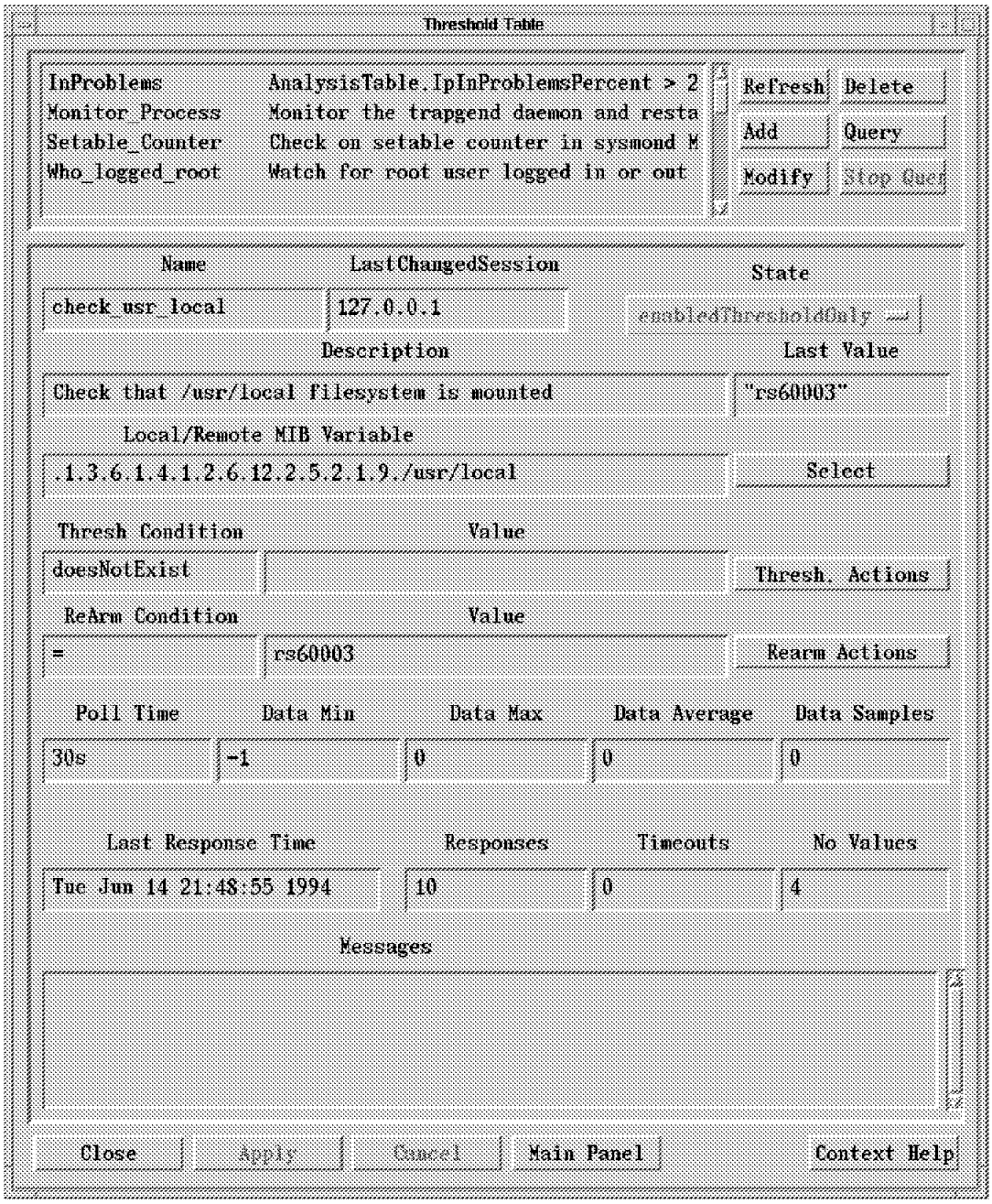


Figure 141. Systems Monitor Threshold Table Definition for NFS Monitoring

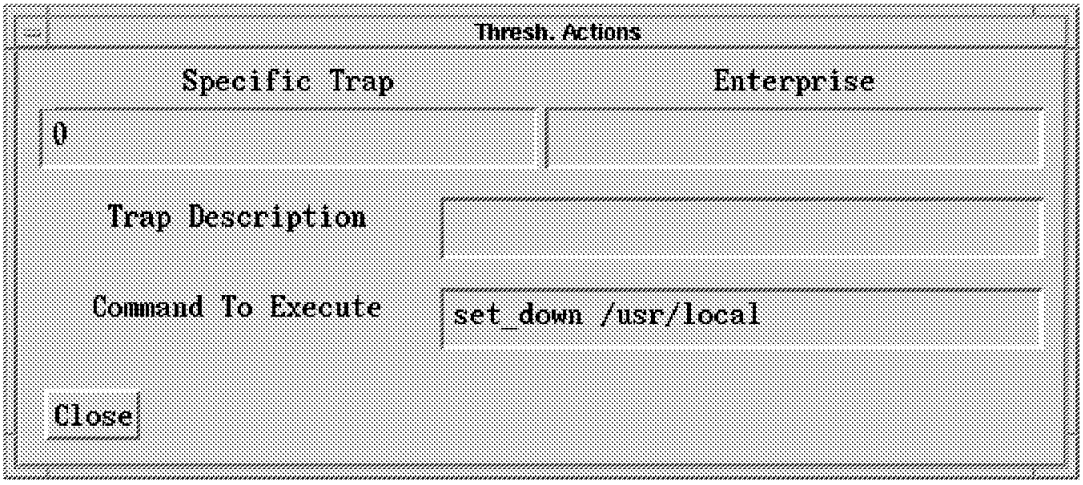


Figure 142. Systems Monitor Threshold Action Definition. This action is triggered when the threshold defined in the previous figure is exceeded.

The set_down and set_up shell scripts each contain a single snmptrap command. This is set_down:

```
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 143. 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 map to different NetView for AIX symbol statuses and, therefore, colors. The possible combinations are listed in A.3.3, “Mapping States and Status to NetView for AIX Displays” on page 249.

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.

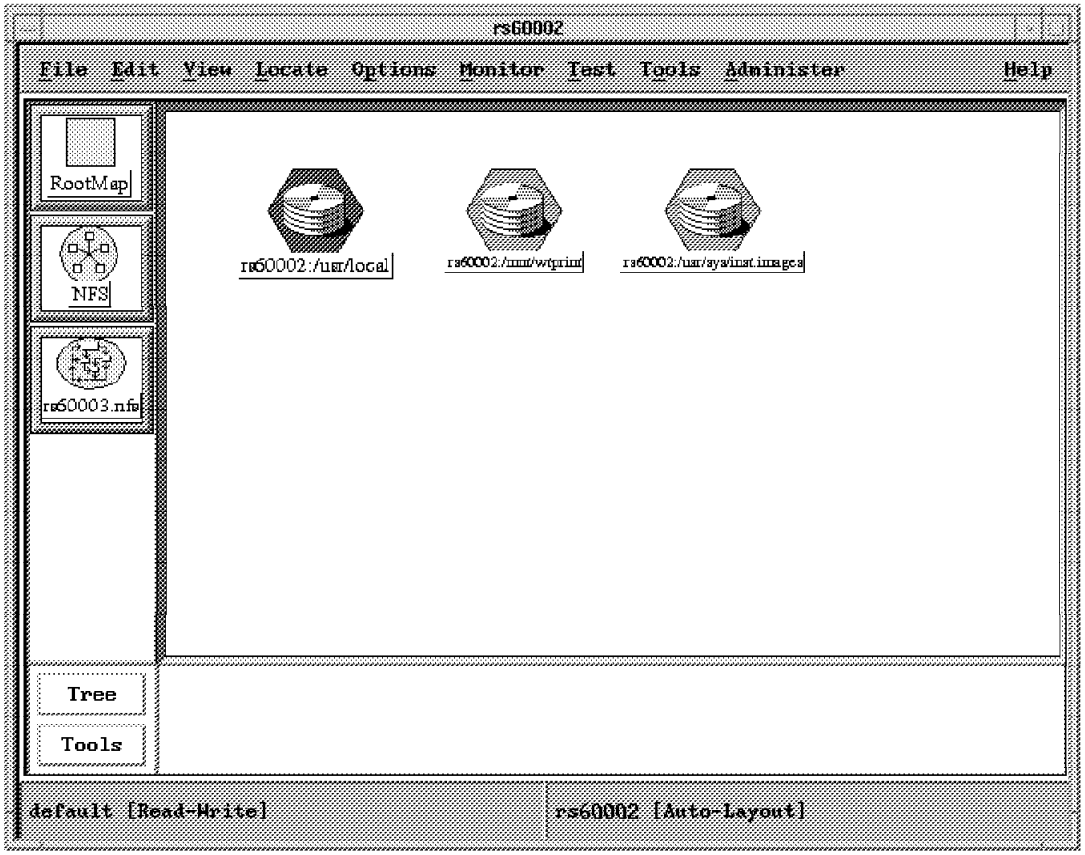


Figure 144. File Systems Submap with Status Change

5.7.1.6 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 145. 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 146 on page 192.

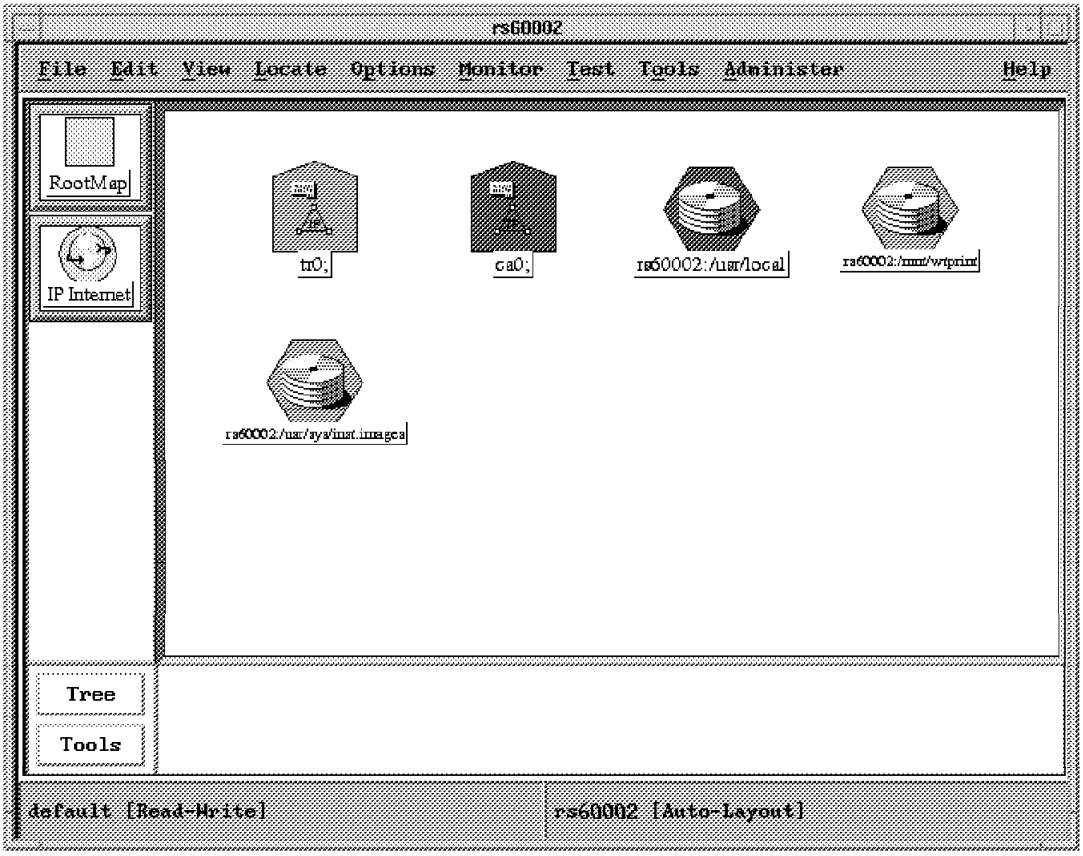


Figure 146. Merged Lowest-Layer Submap Due to SAP Correlation. Notice that both the NFS file systems and the IP interfaces can now be seen to be "part of" rs60002. Status changes to any of them may be propagated up both submap trees.

Another feature provided by SAP correlation is the protocol switching function. Under the **Views** menu bar there is a **Protocols** menu option:

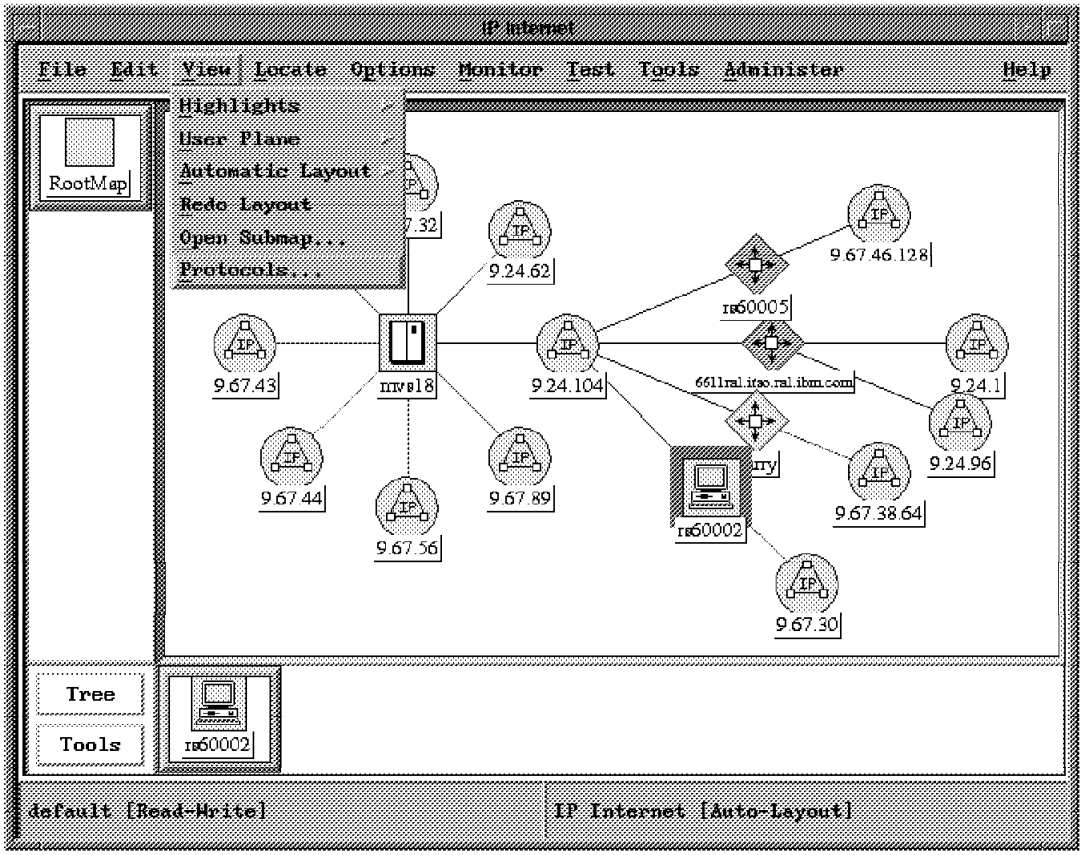


Figure 147. Protocol Switching Option

If we select rs60002 and select the **Protocols** option we see the following panel (Figure 148 on page 193) from which we can directly pass to the different worlds in which rs60002 exists, in this case IP or NFS.

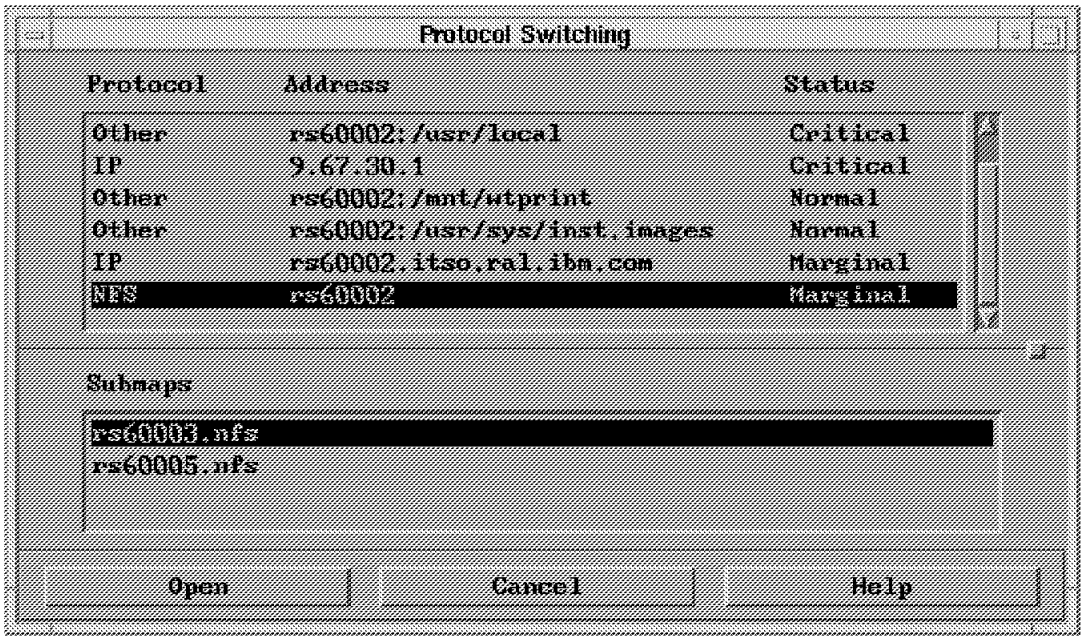


Figure 148. Protocol Switching Panel

Chapter 6. Manager Takeover

One of the new features of NetView for AIX V3R1 is the *Manager Take-over* function. This feature allows the management of the IP network to be split up into segments, with multiple copies of NetView for AIX V3R1 each managing its own defined resources. These separated segments are called Spheres Of Control (SOC). Devices outside the SOC of a particular manager will be *unmanaged*. That is, no polling for status or configuration will be performed.

6.1 Definitions

- We use the term *object* to refer to a managed device or collection of devices in an IP network. For example, a RISC System/6000 or 6611 router, would be objects, as would an IP subnetwork or segment.
- A *Container* must be an Internet, Network, Location or Segment object. That is, a collection of objects linked objects grouped together and managed by one or more manager nodes.
- Each copy of NetView for AIX V3R1 can be configured as a *Manager station* and/or a *Backup manager station* for some containers.
- Managers and Containers must exist as defined objects in the NetView for AIX database.
- Each of the manager nodes will check the status of the other manager nodes on the network. When one of the manager nodes becomes inactive, a message box is displayed to notify the operator. On the backup manager node, the container objects that were within the SOC of the failing manager become managed. A separate submap is displayed to show the objects located in the newly-managed containers.

  Depending on the configuration, the backup manager does not have to monitor all the containers, it may choose to monitor just critical ones for example.
- When the inactive manager becomes active again, another message box will appear, and the reversal takes place. The container objects that were taken over by the backup manager will become unmanaged, once the operator has closed the submaps they are in.

  The objects do not become unmanaged immediately because the backup manager may be performing tasks on them.

6.1.1 Management Example Scenario

In Figure 149 on page 196, an *Area* is defined as a group of Containers, that are in the Sphere of Control (SOC) of a manager.

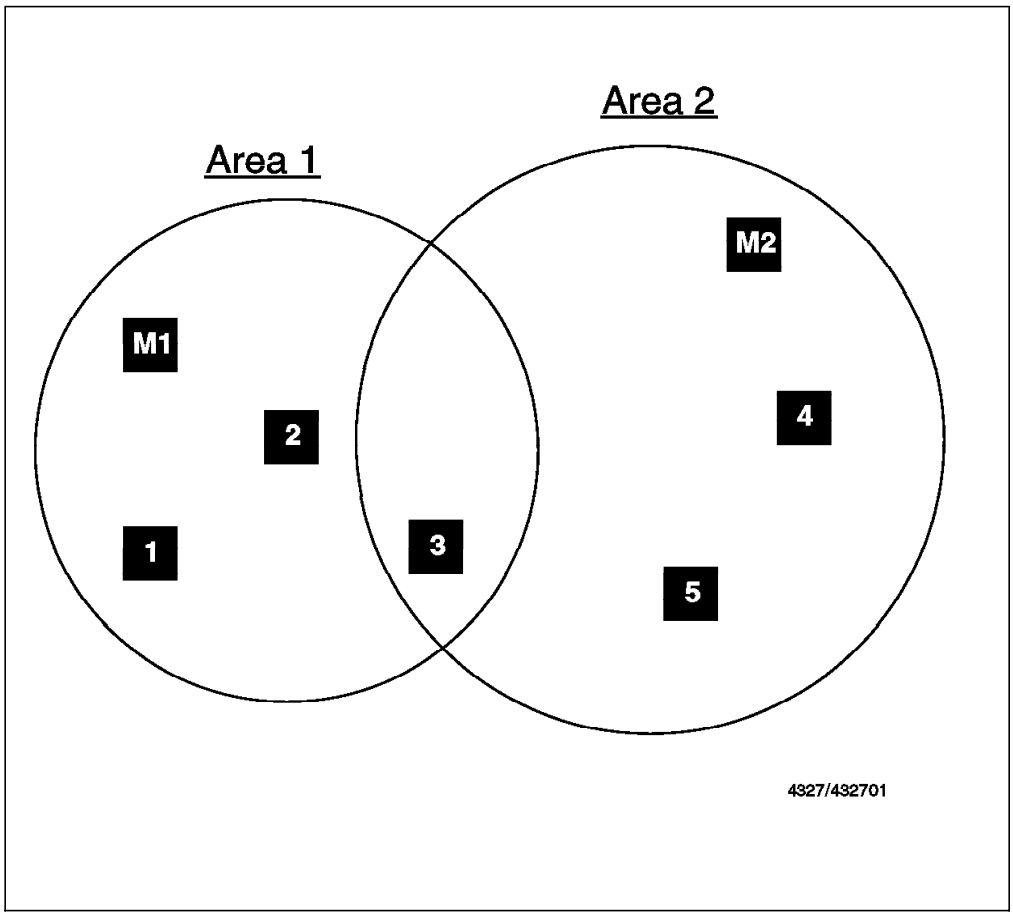


Figure 149. A possible SOC configuration

Manager function can be provided by either AIX NetView/6000 V2R1, or NetView for AIX V3R1, but the backup manager function can only be provided by NetView for AIX V3R1.

M1′s Sphere of Control comprises of Containers 1, 2 and 3 in Area 1, and M2′s comprises Containers 3, 4 and 5 in Area2. In this example, note how M1 or M2 are *both* managing Container 3.

Assuming that both M1 and M2 are running NetView for AIX V3R1, they can each provide backup management for the containers in the other manager′s SOC. For example, we may define that M2 is the backup manager for Containers 1 and 2.

So how does M2 know when M1 goes down, and that M2 should take over the management of Containers 1 and 2? The Netmon daemon in M2 keeps track of the status of the M1 node. When M2 fails to poll M1, the Netmon daemon in M2 issues a *Node Down trap* for M1.

On the M2 machine, a window will be opened for each Container that was being managed by M1. In our example, you will see submaps for Containers 1 and 2. You will also notice that Containers 1 and 2 on the M2 IP map will change from Unmanaged to Managed.

Container 3 was already being managed by M2, so you will see no change of state for Container 3 on the M2 IP map, even if M2 is defined as a backup manager for it.

When M1 returns to the network, M2 will successfully poll M1 and generate a *Node Up* event. M1 has now resumed the management of Containers 1, 2 and 3. So at this point, both M1 and M2 are managing Containers 1, 2 and 3.

M2 users who are still working with Containers 1 and 2, are prompted with a message to close all submaps for Containers 1 and 2 when they have finished working on them. This is done with the ″Manager Restored″ operator prompt that you see in Figure 150.

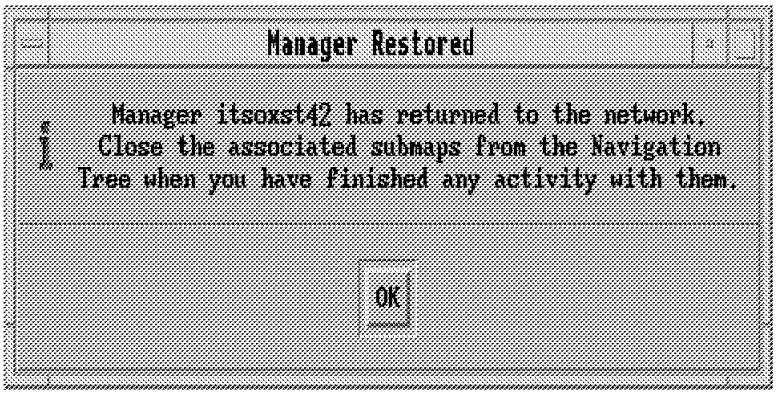


Figure 150. Manager Restored Operator Prompt

The submaps in question *must* be closed from the navigation tree! When this has been done, then the status of Containers 1 and 2, will change from Managed back to Unmanaged.

NOTE:

Notice that the takeover activity involves changing status of nodes from Unmanaged to Managed and back again. It does *not* cause the network discovery routines to be activated. This means that each of the manager nodes must discover the full network before any management containers are defined.

6.2 Configuring Managers and Containers

To configure a node as a manager and define the containers they will manage do one of the following:

1. Define and create a seed file,

   or

2. Use the Backup and Configuration option from the NetView for AIX V3R1 GUI.

As we have described above, the management takeover function relies on one manager polling another. There is no direct manager-to-manager communication. Therefore when a seed file is created it must be distributed to all the relevant management nodes. There are a number of ways to define the SOC for each manager.

Example for Generating the Seed File

A seed file may be created manually, using a text editor. Later in this chapter we show an example of a more automated way to build the seed file by writing a simple NetView for AIX V3R1 application. You may want to read this section before continuing (see 6.2.5, “Creating a Seed File Using a NetView for AIX V3R1 Application” on page 199).

To assist in the design of the SOC and manager nodes on the IP network the use of a planning form is recommended. An example of the form is shown in Table 14.

6.2.1 Aids to Planning the Network Management Topology

Depending on the size and complexity of the managed network, defining the takeover structure may be a complicated procedure. This section details some useful methods to gain the relevant information before commencing.

The first time the Backup configurator is called from NetView for AIX V3R1 it will display a list of containers as defined in the database. (See Figure 155 on page 204) These values (objects) can be added to column one of the worksheet. Alternatively you can build up a list of potential containers from the command line by executing the commands:

```
ovobjprint
ovtopodump
```

These commands will list *selection names* and topology information from the NetView for AIX V3R1 object database.

Table 14. Planning Sheet for what Container is Managed by what Manager

Container	Managed by	Backup managed by
9.24.104	RS60001, RS60003	
9.67.32.64	RS60002	RS60003
9.67.38.64	RS60003	RS60002

Once all the containers have been selected the next task is to decide which NetView for AIX V3R1 nodes are to be managers and backup managers for each container, and to enter the information into the table.

6.2.2 Defining a Seed File

The Seed File is used to determine which nodes are managers and what the relationships and associations are with each container. The seed file itself is composed of three columns:

```
The columns are headed:

Active manager        Container      Backup Manager

Each entry in this file must be enclosed with double-quotes, for
example

"rs60002.itso.ral.ibm.com"   "Segment1" "rs60003.itso.ral.ibm.com"
"rs60002.itso.ral.ibm.com"   "9.20.30"  "rs60003.itso.ral.ibm.com"
```

This example is specifying that the machine rs60002 will manage both Segment1 and 9.20.30. The backup machine will be rs60003 for both containers.

6.2.3 Creating a Seed File

It is critical to the management process that the seed file is defined accurately.

You can create a Seed File with an editor such as vi. Also you can create one from the information that exists in the topology database. Here is a command that will generate a section of a seed file by obtaining objects defined as gateways from the object database and re-directing them to a text file.

```
ovtopodump -lr | \awk '/^HOSTNAME:/ {hostname=$2}
/^FLAGS:.*GATEWAY/ {print hostname}' > seed_file
```

The script will create a file called **seed_file** in the current directory.

Now you may want to include all the objects defined as networks or segments, these are the most likely elements to define as containers. This can be done using the following command:

```
ovtopodump -lr | grep "NETWORK NAME" | cut -d" " -f3
>> seed_file
```

This file now contains all the Network and gateway names for devices in the current network. You can now add any other containers as required. This file must now be modified to insert the names of the managers and the Backup manager(s).

A main problem with this approach is that if you have a large network then the output from these commands may be quite large and it will be quite difficult to relate the information with the NetView for AIX V3R1 IP map.

6.2.4 Using the Seed File with the NetView for AIX

After you determined and created a seed file you must pass it to the backup process to allow to the NetView for AIX to use it.

The seed file is passed to the backup process by modifying the command line for the backup process in the application registration file, */usr/OV/registration/C/backup*.

Edit this file and find the line:

```
Command -Shared -Initial -Restart "${BackupDir:-/usr/OV/bin}/backup";
```

Add the name of the seed file as follows:

```
backup `-s /path/seed_file_name`";
```

6.2.5 Creating a Seed File Using a NetView for AIX V3R1 Application

This example consists of a number of 'C' programs, shell scripts and modifications to the NetView for AIX registration files. The objective is to provide a facility to create a seed file using a "point and shoot" approach. Once this configuration has been set up it will allow fast and efficient generation of seed files.

This example shows a real situation, so the reference to specific nodes and networks will have to be tailored to suit different environments.

This list describes the programs created for this automatic seed file generation example:

- build_seedfile.c - Application code called from the NetView for AIX pull-down menus.
- req_seed.ksh - Shell script to prompt for a seed file to run the backup process.
- build_seedfile.ksh - Shell script to prompt for the manager and backup manager node names

The programs were located in the following directories.

`build_seedfile` - `/u/paul/progs/build_seedfile`

`req_seed.ksh` - `/u/paul/progs/req_seed.ksh`

`build_seed.ksh` - `/u/paul/progs/build_seed.ksh`

These programs are listed in Appendix B, “Automatic Seed File Example Programs” on page 251.

The process we followed is described below:

- Modify the backup registration file `/usr/OV/registration/C/backup` to read as shown in Figure 181 on page 251. This defines menu entries under the Administer->Backup path and also specifies that the `backup` daemon is started automatically when NetView for AIX V3R1 is started. The only change we have made to this is to add the ″Build Seedfile″ entry and its associated `Action` definition.
- Re-start NetView for AIX to incorporate the changes made.
- Open the IP-Internet Map shown in Figure 151 on page 201.

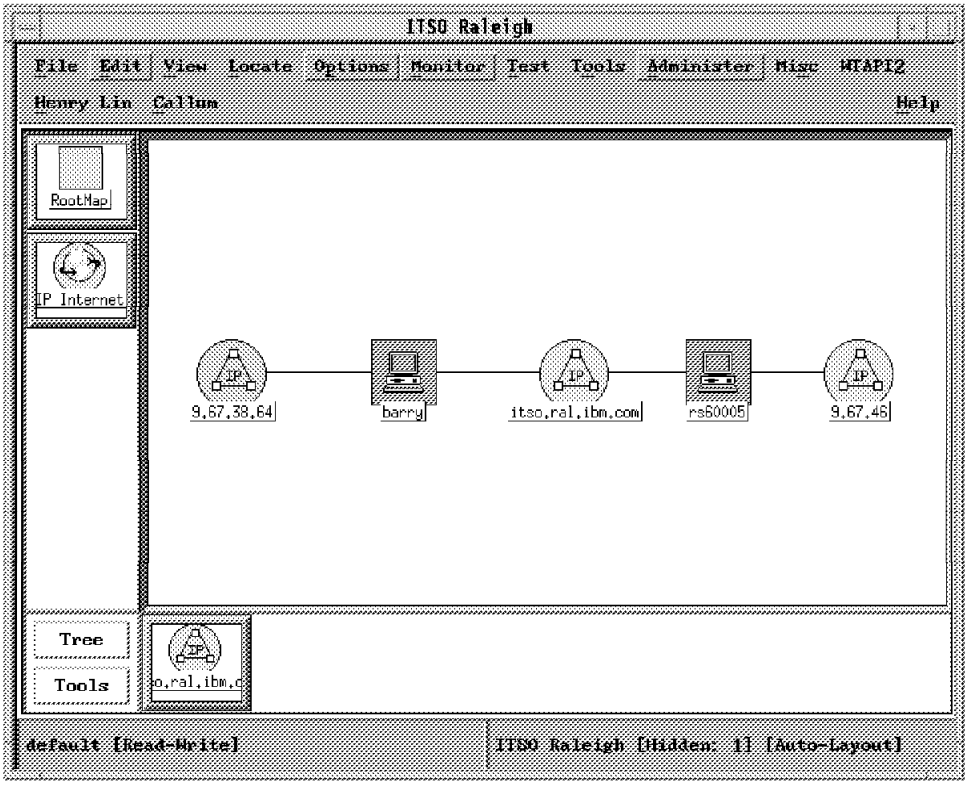


Figure 151. ITSO Raleigh Network IP Map

- Select the two Segments **itso.ral.ibm.com** and **9.67.46** (These will be the containers)
- From the pull-down menu: choose our new option:**->Administer->Backup->Build Seed File**. This will open the window Figure 152 on page 202

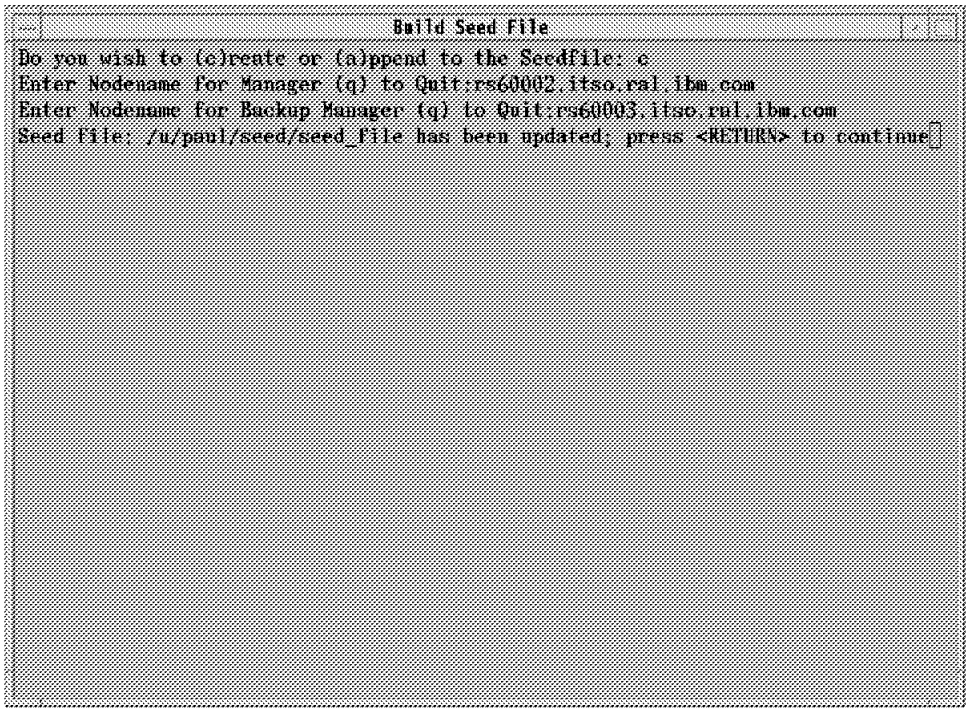


Figure 152. Build Seed File

- Enter the selection name for the manager and backup manager nodes for the selected containers (the selection name is the name by which the manager is known in the NetView for AIX V3R1 object database. As we are using DNS, this is the fully-qualified domain name).
- Select the internet icon **9.67.38.64**
- From the pull-down menu choose **->Administer->Backup->Build**
- Enter (a) to append, and rs60003 as the manager and rs60002 as the backup.
- The seed file generated by this sequence of actions is shown below:

```
"rs60002.itso.ral.ibm.com" "itso.ral.ibm.com" "rs60003.itso.ral.ibm.com"
"rs60002.itso.ral.ibm.com" "9.67.46"          "rs60003.itso.ral.ibm.com"
"rs60003.itso.ral.ibm.com" "9.67.38.64"       "rs60002.itso.ral.ibm.com
```

Figure 153. Example Seed File

- From the pull-down menu choose **Administer->Backup->Read Seedfile** (See Figure 154 on page 203).
- Enter the name of the seed file. (/u/paul/seed/seed_file)

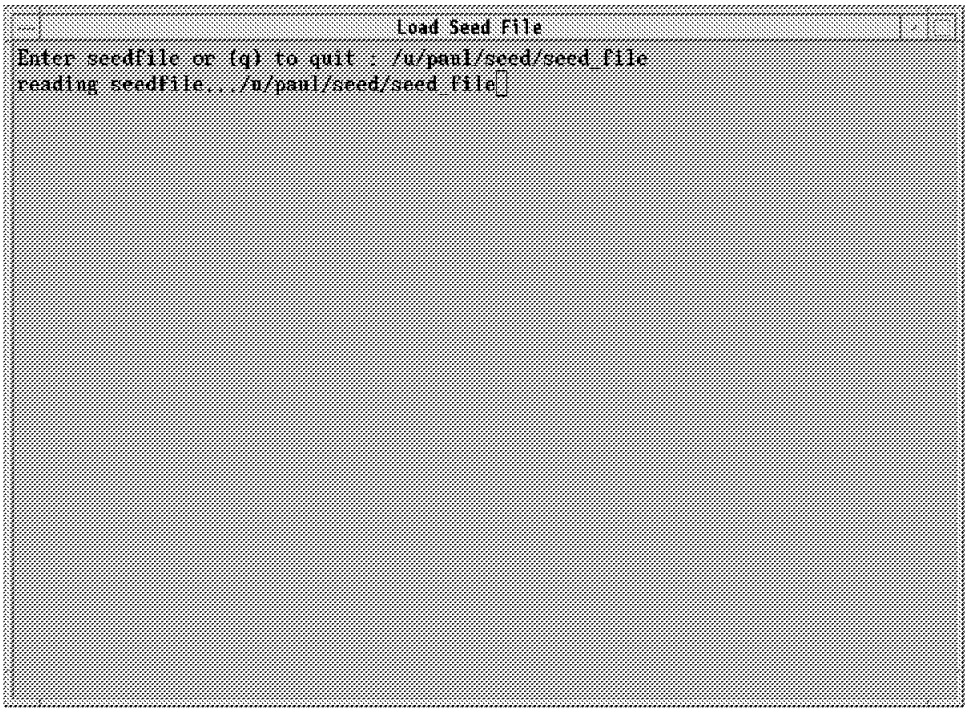


Figure 154. Read Seed File

Different Seed Files

Although we only used Read Seed File for testing purposes, there may be examples where you want to use the facility in production. For example there may be a requirement to run different seed files on a regular basis for management control at differing times of the day, for example, one version of NetView for AIX V3R1 for working hours and another for night time cover. In such a case the script would have to be modified to kill the existing backup routine and restart with the new configuration.

- The backup process has now started for this node.

To check the configuration do the following:

- From the pull-down menu select **Administer->Backup->Backup Configuration**.
- From this screen (Figure 155 on page 204) select **rs60002.itso.ral.ibm.com**.
- Check the information on the screen (Figure 156 on page 204).

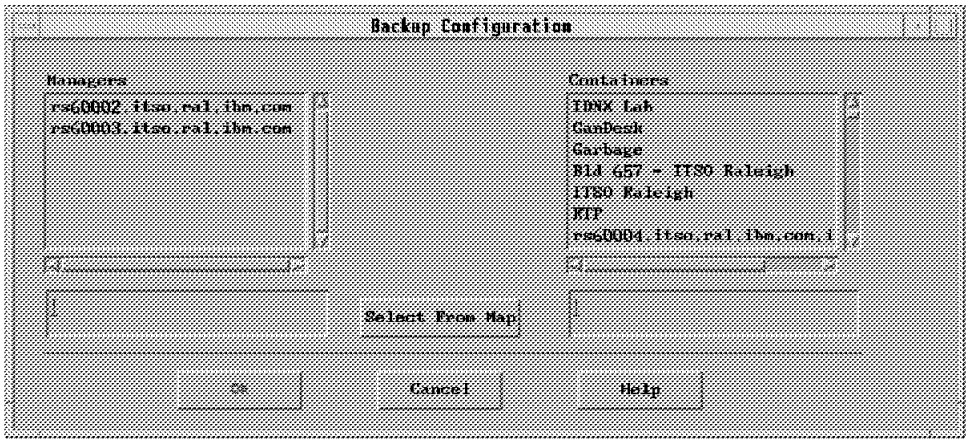


Figure 155. Initial Backup Configuration Screen

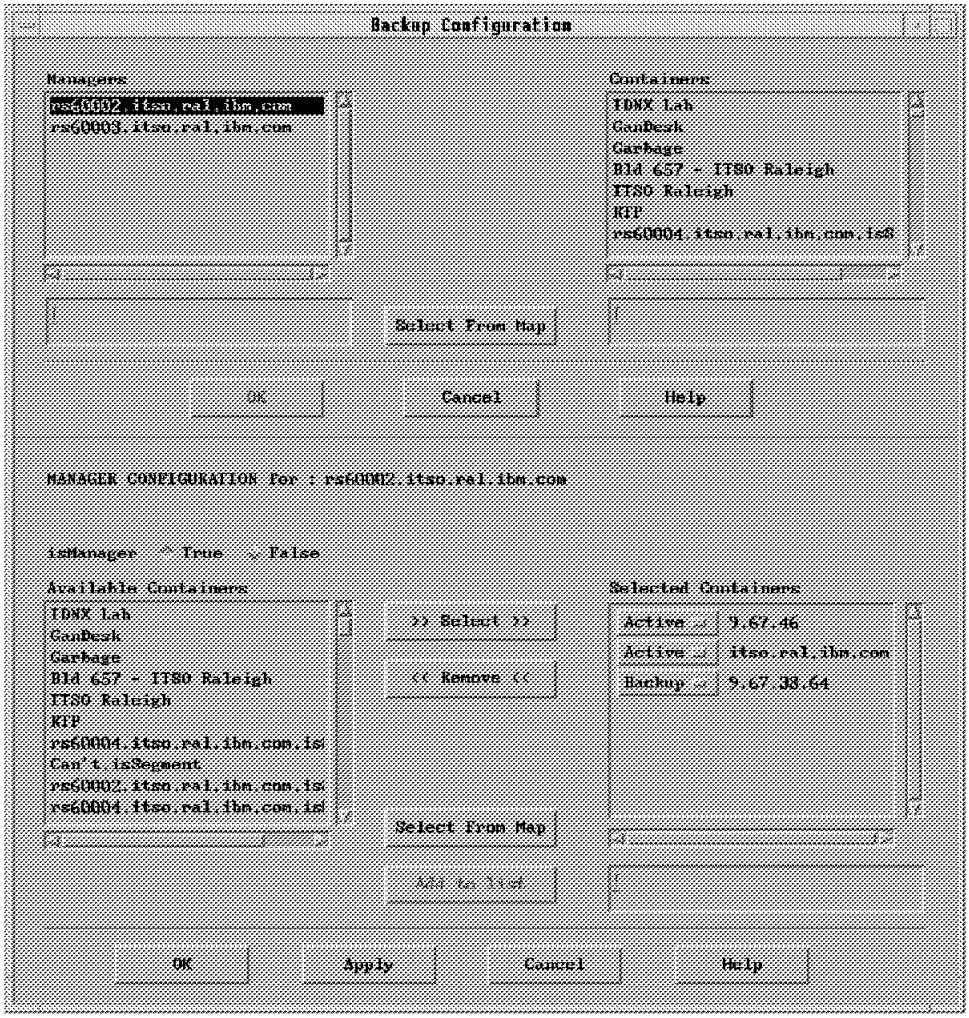


Figure 156. Complete Backup Configuration Screen

The configuration shown is the manager configuration settings for the node rs60002. The window titled ″Selected Containers″ shows which of the containers are actively being managed and those which this node is a backup manager for (for example, the ones chosen from the NetView for AIX V3R1 GUI).

There should be no further need to execute the build seed file option again. Once the data has been entered in the NetView for AIX database it will always appear.

6.3 Running the Backup Process

The NetView for AIX daemon that initiates the management takeover process is called *backup*; there is no relation to the standard AIX command by the same name. Always supply the full path name of this command `/usr/OV/bin/backup`.

If there is a requirement to change the management associations after the seed file is used (for example, adding one new container to a particular manager) then this should be done using the Backup Configurator described in 6.4, "Backup Configuration EUI" on page 205.

If an association has been made in error, or a configuration change required, then this is achieved by using the Backup Configurator to remove the selected containers for a particular manager and changing the **isManager** option to false.

6.4 Backup Configuration EUI

The Backup Configurator is a graphical user interface application. This can be used by an operator to view and/or modify the configuration of Backup Managers and/or Containers. The Backup Configurator is started from the NetView for AIX V3R1 menu bar under: **Administer** → **Backup** → **Backup Configuration**.

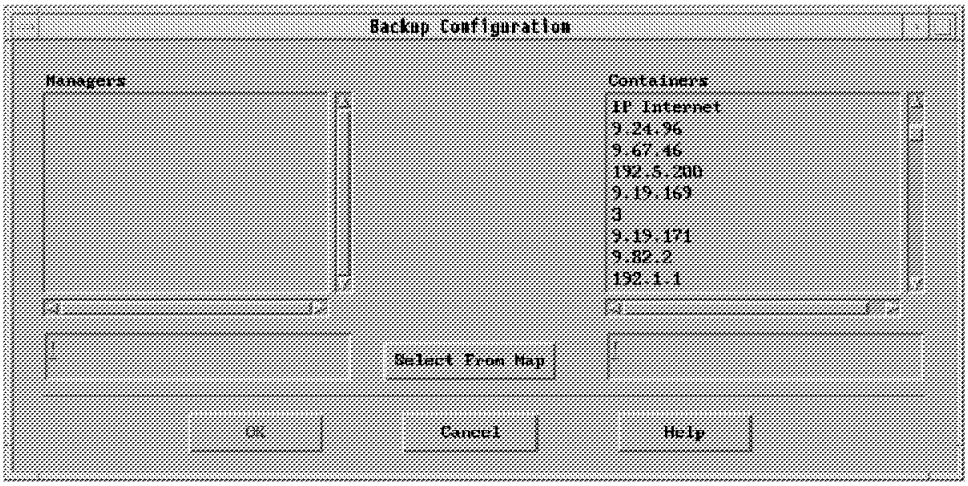


Figure 157. Example of a Backup Configurator Initial Screen

To exit the application, just click on **Cancel**.

The list under the heading Containers shows *all* objects found in the NetView for AIX Database, that match the Internet, Network, Location or Segment object selection criteria. From this initial screen you select either a Manager or a Container to work on.

You must have the current map open in read-write mode to configure the manager.

6.4.1 Adding a New Manager to the Backup Configuration

If the seed file was *not* used when starting the backup process, then no managers should be listed in the Managers selection box. To add a manager, choose one of the following:

1. Click on the **Managers** selection window, and then type in the name of the manager, and then select **OK**.
2. Select a manager on the IP map, and then press the Select from Map button on the Backup Configurator menu.

You are then presented with a screen like that in the example in Figure 158.

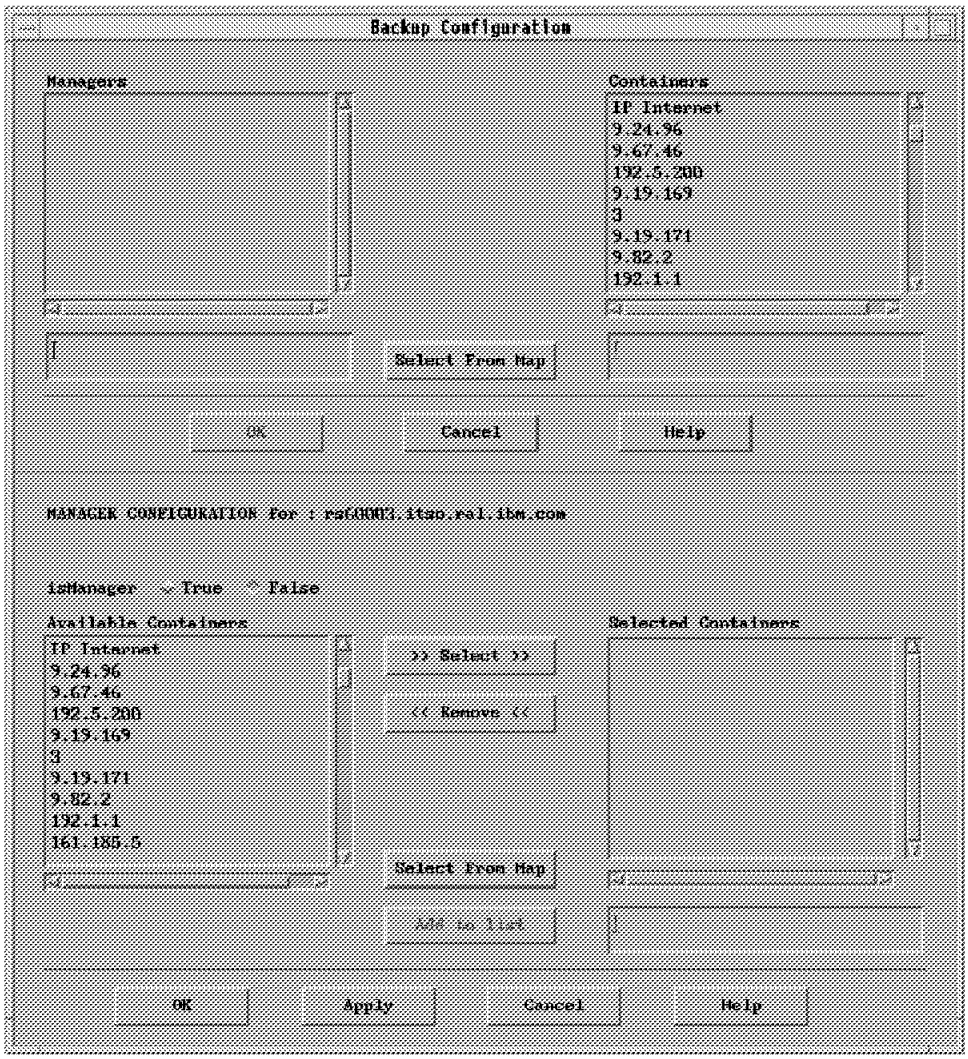


Figure 158. Backup Configurator with RS60003 Selected (isManager=False)

Notice that initially the isManager flag is set to *False*. Change this flag to *True*, and the node is added to the managers list. Changing the flag back to *False* removes the node from the managers list.

When set to *True*, as in the example in Figure 159 on page 207, you cannot select the Containers that you want to be managed by this manager from the Available Containers list.

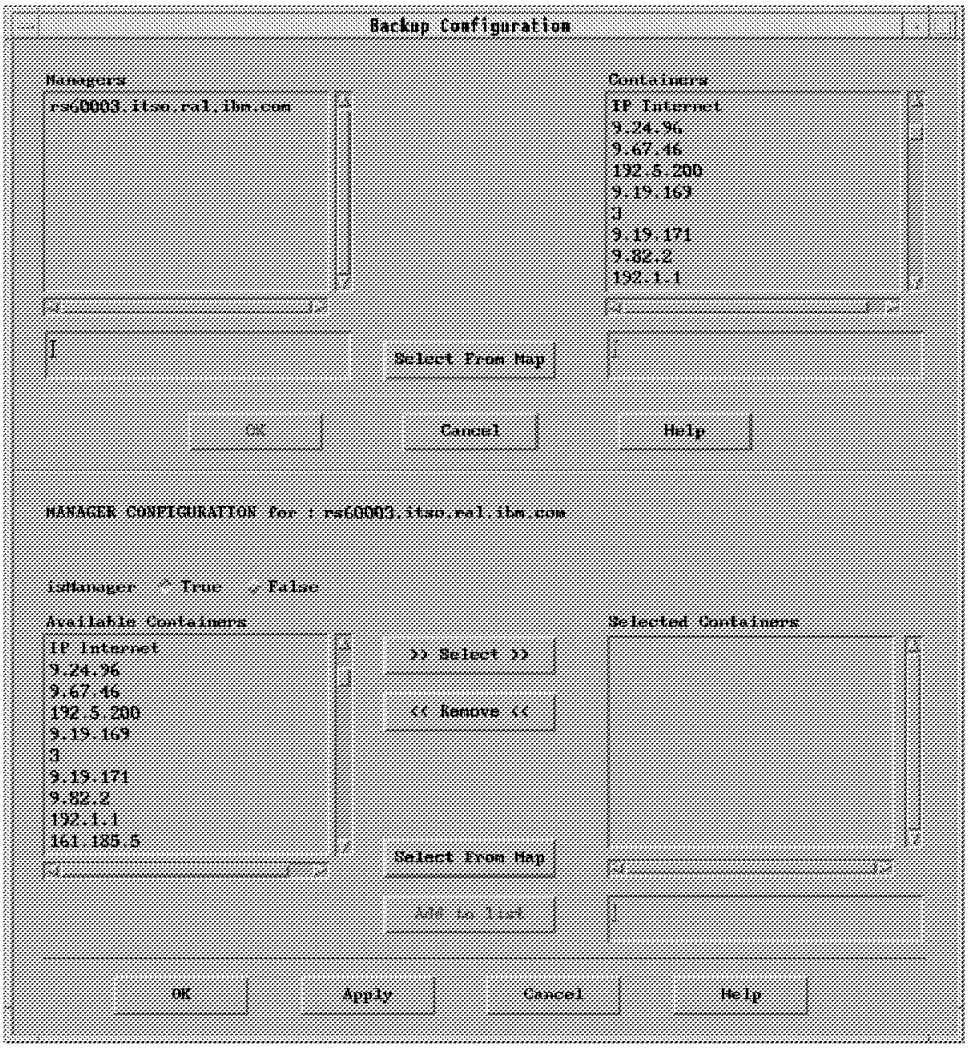


Figure 159. Backup Configurator with RS60003 Selected and isManager=True

When you select a Container, the Container will be added to the Selected Container list. Now you have the possibility for this Manager to be either an Active Manager or a Backup manager, the default here is the Backup association between that Container and this Manager.

Note that this will make no change to the management status of this Container on other NetView/6000 managers.

For specific conditions on changing colors (or changing states) of Containers on the Maps, see Table 15 on page 208.

Table 15. Seen from RS60003, Colors of Container on IP Internet Map			
Container	**Managed by RS60002 (remote)**	**Managed by RS60003 (local)**	**State on IP Internet map**
9.67.46	N	N	Managed
9.67.46	Y	N	Unmanaged
9.67.46	Y	Y	Managed
9.67.46	N	Y	Managed
9.67.46	Y	Y	Managed

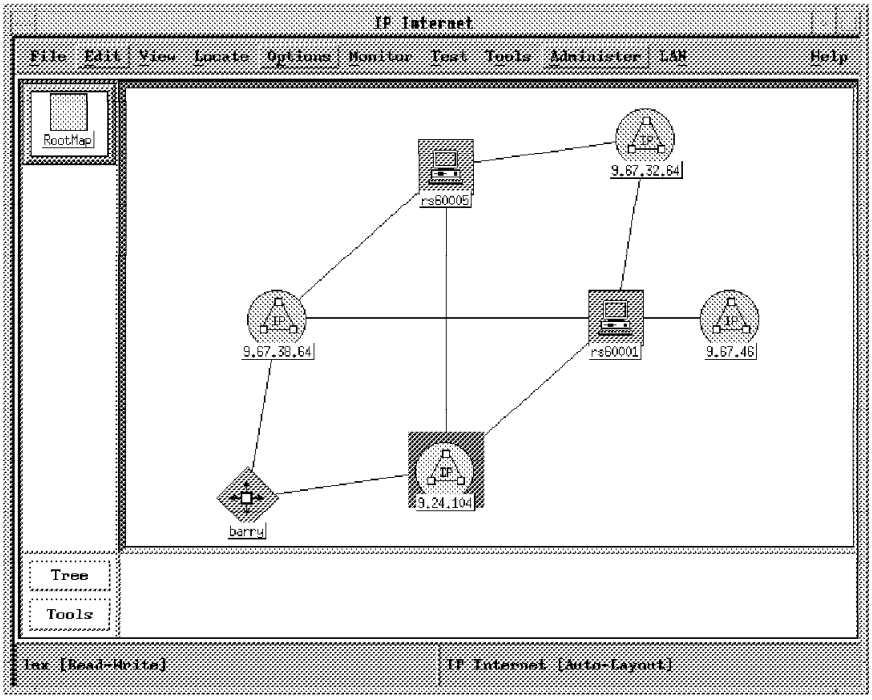


Figure 160. Highlighted Symbol is the Only One managed

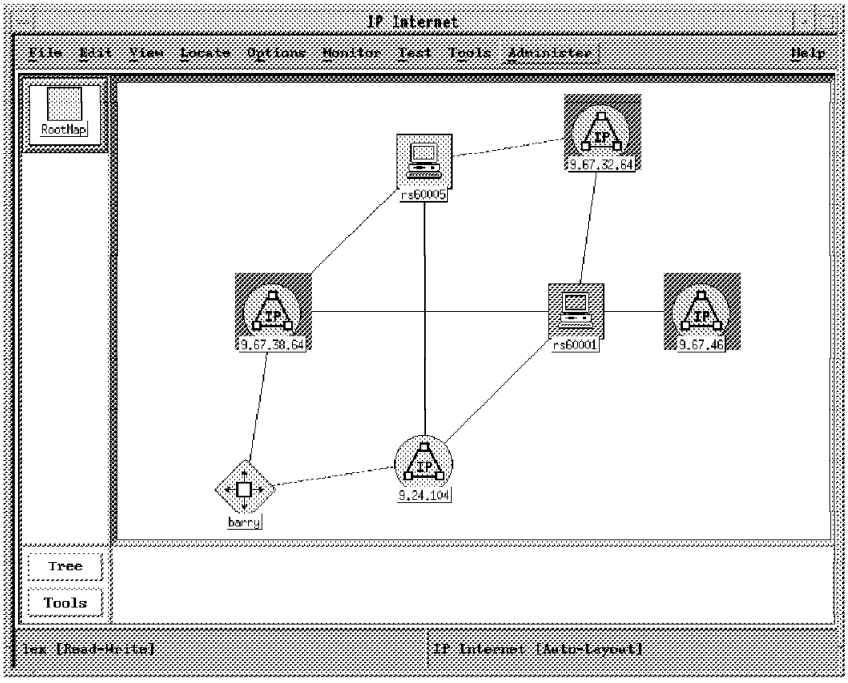


Figure 161. Highlighted Symbols Have Changed Color

If you are configuring a remote manager, for example a NetView for AIX V3R1 management station other than yourself, then the status of the Container on the local IP submap will not change.

If you are adding a Container to the local manager, then it will only become managed when it has been removed from the Active Container list for the remote manager.

Remember that the configuration information is held locally by the machine running the configuration utility, so what is important is what the local manager perceives to be the configuration of the remote manager. It might be that the configuration of the remote manager shows a different picture. The result is that it is important that the distribution of backup configuration information is coordinated among NetView for AIX V3R1 managers.

When you change the isManager flag back to true, you also get a message saying that this manager has returned to the network, as in Figure 150 on page 197.

For some other examples, see Figure 162 on page 210.

6.4.2 Container Configuration

If you select a Container from the Containers list, you can select the manager or managers that should manage this Container. If this configuration has already been done, then you will get a list of the managers managing this Container.

6.4.3 Configuration Summary

1. You can configure the manager backup configuration using the Backup Configurator EUI. Use this only for local changes; otherwise use a seed file instead, and distribute it among all of the managers in the backup configuration.
2. The configuration can be done from the perspective of the manager, or from that of the Container.
3. The configuration information is held *locally* to the manager doing the configuration process. There is no manager-to-manager communication.

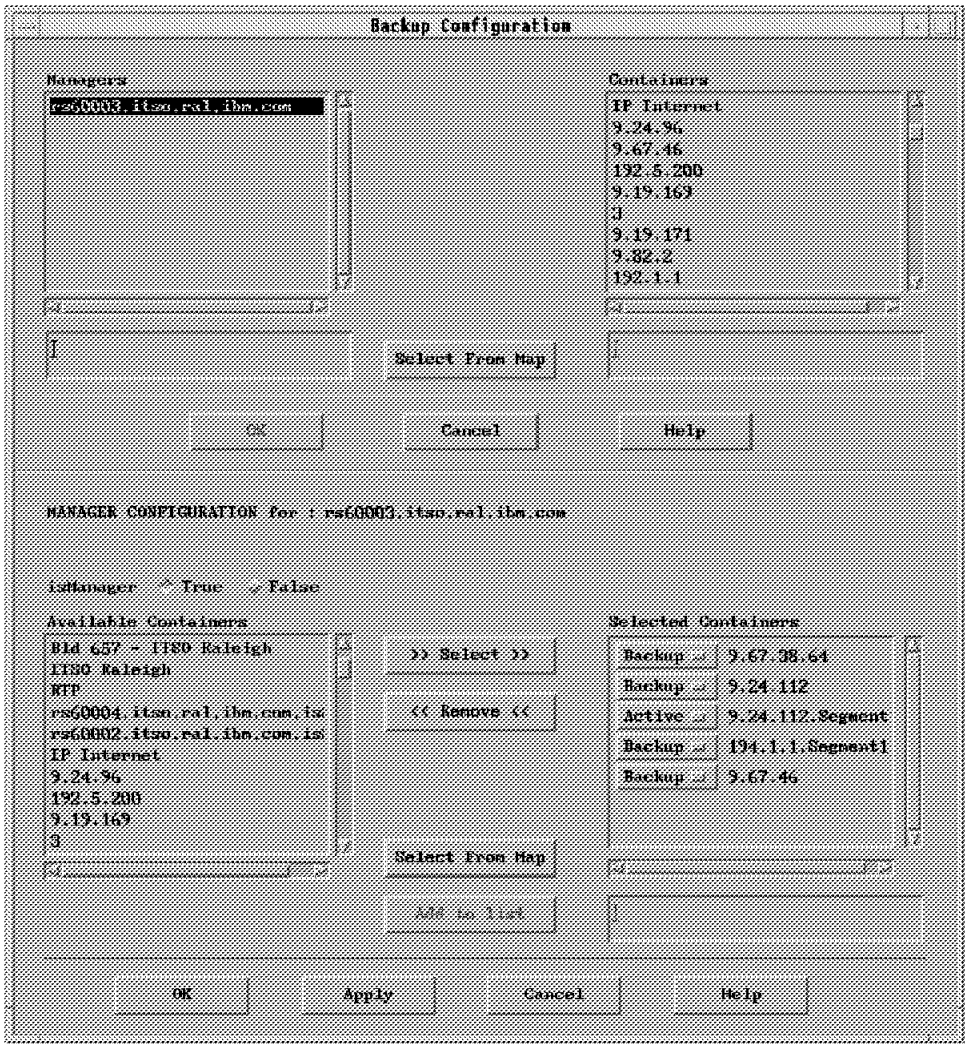


Figure 162. Backup Configurator with RS60003 Managing Some Containers

6.5 The ITSO Environment

In the example scenario there are three machines included in the takeover configuration. These are:

1. RS60001 running NetView for AIX V3R1
2. RS60002 running AIX NetView/6000 V2R1

3. RS60003 running NetView for AIX V3R1

The following shows two manager-failure scenarios:

1. RS60001 and RS60003 - both running NetView for AIX V3R1
2. As in item 1, but now including RS60002 which is running an old version of AIX NetView/6000

For all our testing we used shell scripts to raise the relevant Node Up and Node Down events. The manager takeover process relies on these events to trigger all backup and return functions. Normally Node Up and Node Down are generated internally by the netmon daemon, as a result of its regular polling cycle. This implies that a manager failure is detected only when the machine on which NetView for AIX is running fails. However there are some situations in which the machine may still be running, but we still wish for takeover to happen, for example:

- When testing (as in our case)
- When carrying out maintenance on the machine
- When backing up or re-configuring the machine

In these cases it is useful to be able to trigger manager takeover without having to take the machine on which the failing manager runs offline.

Our example extends the takeover process so that the the status a critical NetView for AIX process is monitored. We check that the netmon daemon is running. Without netmon, NetView for AIX will not perform any network polling, so its existence is a good measure of the health of a NetView for AIX system.

The shell script shown in Figure 163 on page 212 is started by NetView for AIX V3R1 after netmon has been initiated. If netmon fails, then the script generates an SNMP TRAP advising all hosts named in `/etc/snmpd.conf` of a node down condition.

When netmon is restarted, it sends an SNMP TRAP advising of a node up condition.

aixterm

```
#!/usr/bin/ksh
USAGE="netmtst.ksh" #Check if NETMON is running
while :
 do
  VAR=`ovstatus netmon | tail -6 | awk '{print $2}'`
  echo $VAR  | grep NOT_RUNNING
   if  [ "$?" -ne 0 ]
    then
     sleep 60
    else
     echo "Netmon went down."
     "/u/lex/ndlex.awk"
     while :
      do
       VAR=`ovstatus netmon | tail -6 | awk '{print $2}'`
       echo $VAR  | grep RUNNING
       if  [ "$?" -ne 0 ]
        then
         sleep 60
        else
         echo "Netmon came up."
         "/u/lex/nulex.awk"
         break
       fi
      done
   fi
 done
```

Figure 163. A Shell Script to Test for a Running Netmon

The netmtst.ksh script calls ndlex.awk and nulex.awk, and these are shown below.

ndlex.ksh and nulex.ksh

ndlex.awk: awk '$1~ /trap/ {system("ndlex.ksh " $3)}' /etc/snmpd.conf
nulex.awk: awk '$1~ /trap/ {system("nulex.ksh " $3)}' /etc/snmpd.conf

The contents of **ndlex.ksh** and **nulex.ksh** looks like:

ndlex.ksh:

```
#!/bin/ksh
host='hostname'
snmptrap $1 .1.3.6.1.4.1.2.6.3 $host 6 58916865 0 \
        .1.3.6.1.4.1.2.6.3.1.1.2.0 Integer 9 \
        .1.3.6.1.4.1.2.6.3.1.1.3.0 OctetString "LexMol" \
        .1.3.6.1.4.1.2.6.3.1.1.4.0 OctetString "Node Down"
```

nulex.ksh:

```
#!/bin/ksh
host='hostname'
snmptrap $1 .1.3.6.1.4.1.2.6.3 $host 6 58916864 0 \
        .1.3.6.1.4.1.2.6.3.1.1.2.0 Integer 9 \
        .1.3.6.1.4.1.2.6.3.1.1.3.0 OctetString "LexMol" \
        .1.3.6.1.4.1.2.6.3.1.1.4.0 OctetString "Node Up"
```

In order to register this script to NetView for AIX V3R1, the following LRF file should be placed in /usr/OV/lrf:

/usr/OV/lrf/netmtst.lrf

```
netmtst : /u/lex/netmtst.ksh :
OVs_YES_START : Netmon : : OVs_NON_WELL_BEHAVED : :
```

Issue the `ovaddobj netmtst.lrf` command to add this object to the NetView for AIX V3R1 database.

6.5.1 Using Netmon Status to Drive Manager Backup, Case 1

6.5.1.1 Manager Failures with All Managers Running NetView for AIX V3R1

In this sample configuration, RS60002 is the backup for RS60003, and RS60003 is the backup for RS60002.

The manager - Container backup was designed according to the table shown in Table 16.

Table 16. Manager / Container Design for Scenario 1

Container	Managed By	Backup Managed By
9.24.104	RS60001 and RS60003	None
9.67.32.64	RS60001	RS60003
9.67.38.64	RS60003	RS60001

For the design in Table 16, we made the seed file that is shown in Table 17 on page 214. This seed file was distributed to RS60001 and RS60003.

As RS60001 and RS60003 are both within network 9.24.104, both NetView for AIX V3R1s were managing this network.

The following tests were done to see what happened on the NetView for AIX V3R1 of each management station.

Expectations

Whenever a Node down trap is received by NetView for AIX V3R1 for any Manager that has NetView for AIX V3R1, we should get a pop up screen as can be seen in Figure 164 on page 215. Some time later we should get a window created by NetView for AIX V3R1 for each Container where our NetView for AIX V3R1 is a Backup Manager for.

Whenever a Node up trap is received for a NetView for AIX V3R1 we have taken over a resource from, we should get a screen as can be seen in Figure 150 on page 197.

It is not be important how those traps are created. They can be done via snmptrap or done by NetView for AIX V3R1 itself after discovering the down/up situation.

1. Start NetView for AIX V3R1 on both machines with the seed file in Table 17 on page 214.
2. Check the configuration via the Backup Configuration menus.

3. Bring down NetView for AIX V3R1 on RS60001 and observe the changes to NetView for AIX V3R1 on RS60003.
4. Observe the changes to the IP Internet map on RS60003.

Table 17. Seed File Contents of RS60001 and RS60003

Active Manager	Container	Backup Manager(s)
″rs60001.itso.ral.ibm.com″	″9.24.104″	
″rs60001.itso.ral.ibm.com″	″9.67.32.64″	″rs60003.itso.ral.ibm.com″
″rs60003.itso.ral.ibm.com″	″9.24.104″	
″rs60003.itso.ral.ibm.com″	″9.67.38.64″	″rs60001.itso.ral.ibm.com″

6.5.1.2 Observations from Two NetView for AIX V3R1 Managers

1. Machines started OK.
2. Displayed backup configuration as per the seed file.
3. The effect of stopping NetView for AIX V3R1 on RS60001:

   Originally, no change on RS60003.

   This is to be expected as the termination of NetView for AIX V3R1, does not generate a *Node Down* trap.

   This was one of the reasons for creating the shell script (mentioned earlier) for checking the availability of the Netmon application. The other reason is described under 6.5.3, “Effect on RS60003 of the Return of RS60001” on page 215.

   To simulate a NetView Down situation we ran the command:

   ```
   ovstop netmon
   ```

   To revive the netmon daemon we issued the command:

   ```
   ovstart netmon
   ```

   This tested the failing of the NetView for AIX V3R1 daemon.

   After the implementation of these scripts, the results were the same as described under 6.5.2, “Effect on RS60003 of a Re-IPL of RS60001.”

6.5.2 Effect on RS60003 of a Re-IPL of RS60001

A re-IPL of RS60001 generates a *Node Down* trap. When this is done, then any versions of NetView for AIX V3R1 running on RS60003 will have maps created on the desktop for all the Containers that were managed by RS60001, before the node down event. In our example, you see maps created for 9.67.32.64. This Container, was unmanaged on RS60003 before the failure of RS60001, but now they become actively managed on RS60003.

All users of RS60003 received a message to advise them that RS60001.itso.ral.ibm.com is down. An example of that screen is shown in Figure 164 on page 215.

6.5.3 Effect on RS60003 of the Return of RS60001

As soon RS60001 returns, a *Node Up* was received on RS60003, advising that RS60001.itso.ral.ibm.com has returned to the network. An example of the message is shown in Figure 150 on page 197. Originally you do not know if NetView for AIX V3R1 is up again on RS60001. This was the second useful reason for running the netmtst.ksh script.

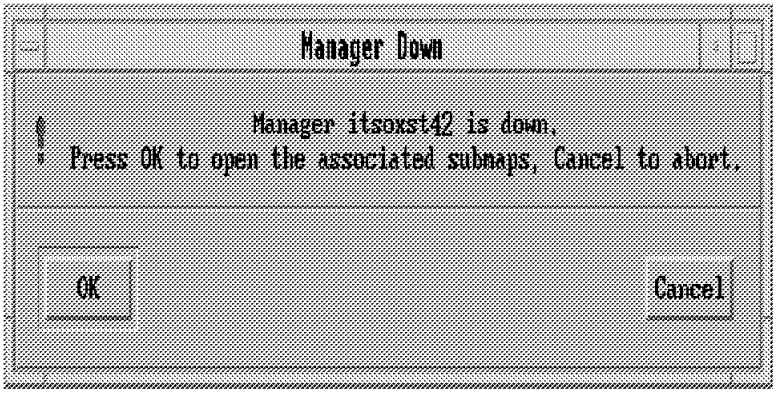


Figure 164. Manager Down Message

At this point, the operators of RS60003 have been told that RS60001 is now managing 9.67.32.64, but we can't be sure that NetView for AIX V3R1 has become active on host RS60001, as we will see the *node up* event before NetView for AIX V3R1 is active.

On the Navigation Tree on RS60003, we then closed the maps related to 9.67.32.64, and observed that 9.67.32.64 became Unmanaged on the RS60003 IP map.

6.5.4 Using Netmon Status to Drive Manager Backup, Case 2

In this case RS60002 is the backup for Container 9.67.32.64.Segment of RS60001 and all of RS60003.

RS60003 is the backup for Container 9.67.32.64 of RS60001 and all of RS60002.

Table 18. Seed File Contents of RS60001, RS60002 and RS60003

Manager	Container
rs60001.itso.ral.ibm.com	9.67.32.64.Segment1
rs60002.itso.ral.ibm.com	9.24.104
rs60002.itso.ral.ibm.com	9.24.104.Segment1
rs60002.itso.ral.ibm.com	9.67.46
rs60002.itso.ral.ibm.com	9.67.46.Segment1
rs60003.itso.ral.ibm.com	9.24.104
rs60003.itso.ral.ibm.com	9.24.104.Segment1
rs60003.itso.ral.ibm.com	9.67.32.64

Just as in Case 1, we installed the netmtst (and related) scripts on NetView machines, to test the running of the Netmon application.

To simulate a NetView Down situation we ran the command:

```
ovstop netmon
```

To restart NetView again we issued the command:

```
ovstart  netmon
```

We saw the takeover of Container 9.67.32.64 by RS60003 and Container 9.67.32.64.Segment1 by RS60002.

When we brought down *RS60002* now, RS60003 took over 9.67.32.64.Segment1, 9.67.46 and 9.67.46.Segment1.

Now we started Netmon on RS60001 again (RS60002) still down).

On RS60003 this gave the message RS60001 came back to the network, so we could close (via the Navigation Tree) 9.67.32.64 and 9.67.32.64.Segment1.

6.6 Usage Notes

The following are a few items to consider:

1. *Do not* Un-manage Backup Managers.

   The local Manager is *not* polling that Backup Manager anymore. In this case it will not see the failure of that Manager, unless you have arranged the nodes as discussed in 6.5, “The ITSO Environment” on page 210. and in Figure 163 on page 212.

2. You can increase the polling rate for remote Managers.
3. In order to see only the remote managers Node-Up an Node_Down traps, you can create a dynamic workspace and filter only the Remote Manager events. This will show the status of the Remote Managers.
4. Do not use names longer then 100 characters.
5. When a Manager takeover has occurred, the traps sent to the Manager that went down are still sent. There is no automatic method for changing the trap destination address in the agents that were sending traps to the just went down Manager. A way to solve this could be one of the following:

   a. Have all agents send traps to all managers.
   b. Make a shell script that changes the trap destination address dynamically of all agents involved.

Chapter 7. wtdriver6/wteuiap6 Sample NetView for AIX EUI API

This chapter summarizes wtdriver6 and its interface with wteuiap6. These are sample NetView for AIX EUI API application programs developed during ITSO residency programs.

In a previous redbook: *Examples of Using AIX NetView/6000 APIs,* GG24-4059, other ITSO-developed sample APIs were discussed. In particular, wtdriver/wteuiap1 and wtdriver2/wteuiap3 were presented. wtdriver6, together with wteuiap6, are replacements for these previous examples.

In addition to additional user functions which are provided by wtdriver6/wteuiap6 over earlier examples, the newer code solves EUI application problems which arise when multiple NetView for AIX operators or maps are active on the same system.

7.1 Summary of NetView for AIX Interfaces

The following figure summarizes interfaces which are available for particular functions and are available for use by NetView for AIX operators and application programs which implement API coding.

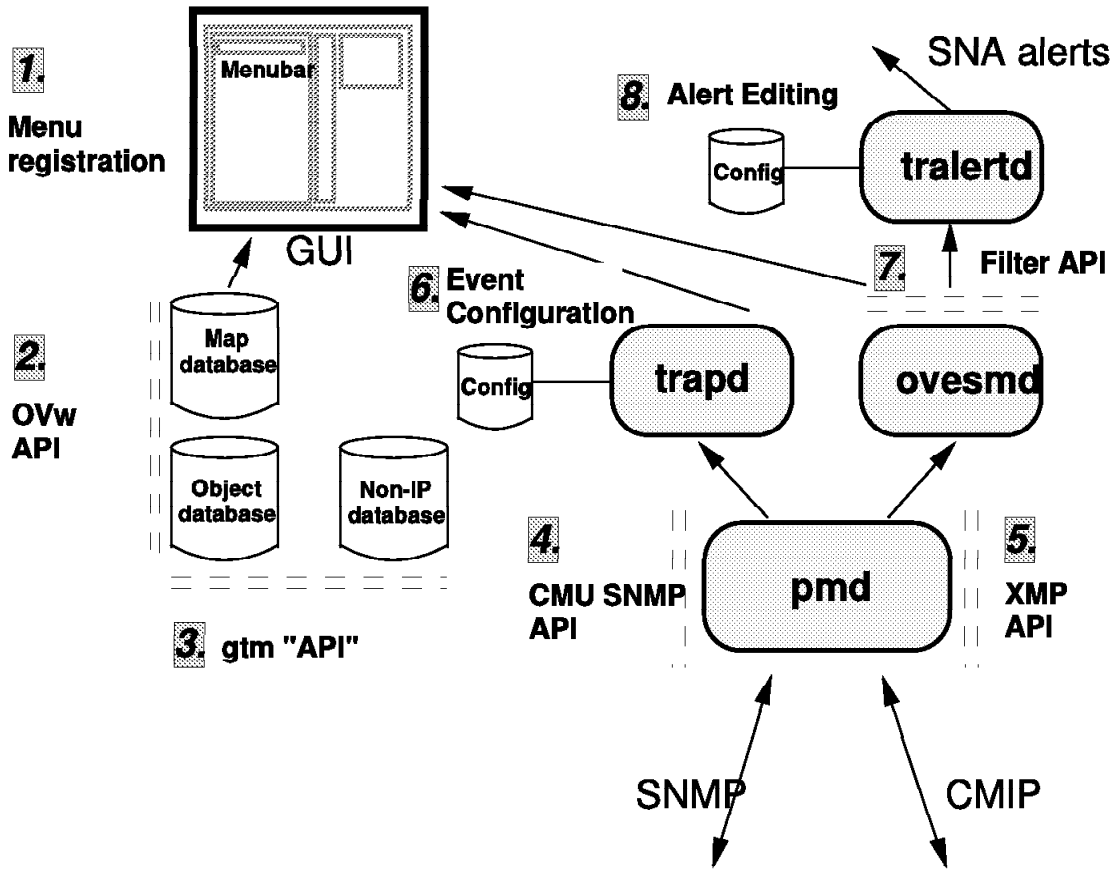


Figure 165. Overview of Possible User Interfaces with NetView for AIX

7.2 ITSO wteuiapx EUI Samples

wtdriverx/wteuiapx are a series of samples giving general-purpose topology manipulation.

Previous to the current wteuiap6, the following general application structure existed.

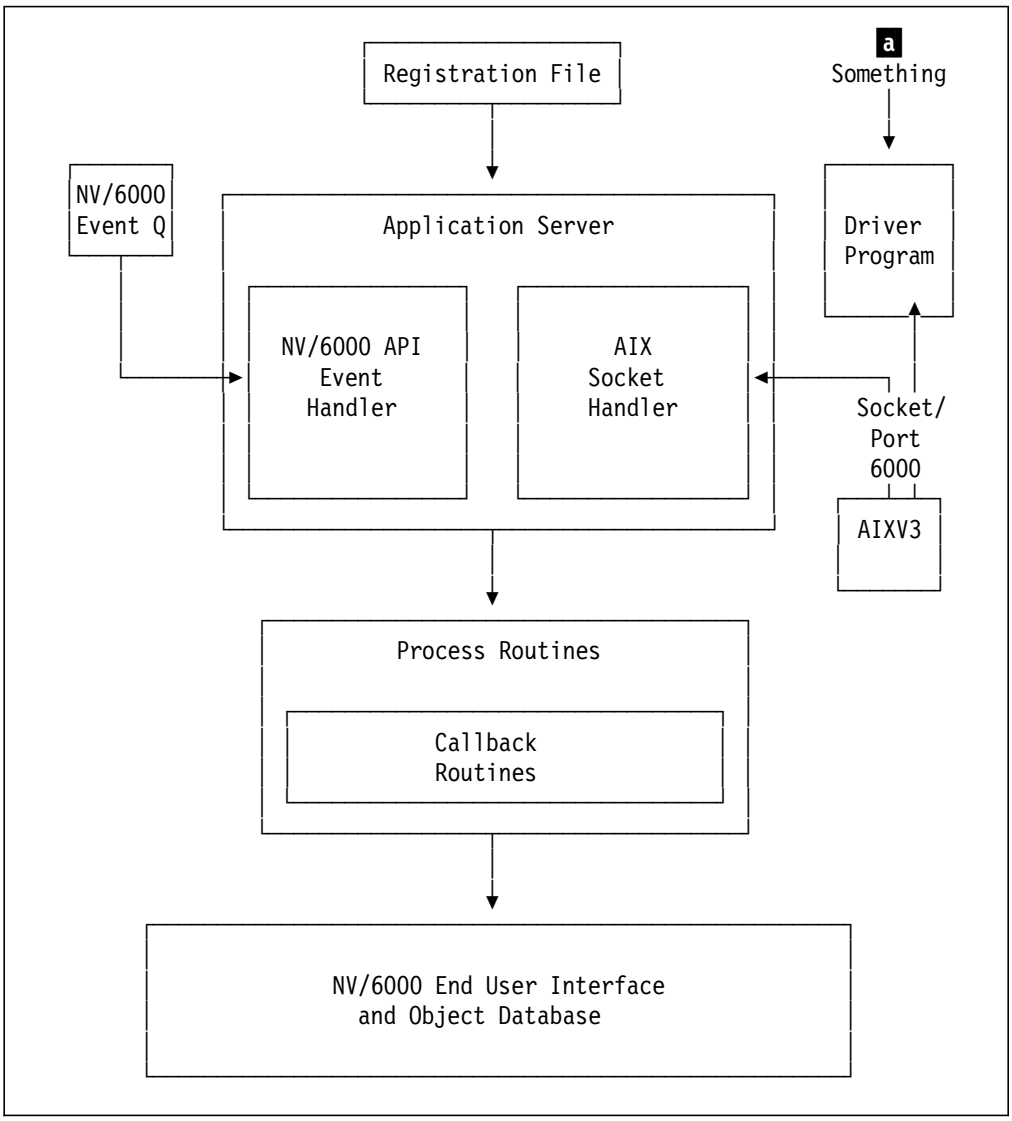


Figure 166. Overview of Example Application wteuiap3

As use of the above general structure evolved, problems were identified, including:

- If multiple operators or multiple NetView for AIX maps were concurrently active in the same NetView for AIX, the previous general structure was insufficient to allow for user access and control of eui functions.
- As multiple applications joined the NetView for AIX family of products and user application needs evolved, requirements such as controlling multiple (base/root) maps related to a particular application and the need for ″merged views″ of submaps required a restructuring of the ITSO sample EUI application.

wtdriver6/wteuiap6 address these problems.

Hereafter, for convenience, we will refer to this application as wteuiap6; however, keep in mind that the user interface to wteuiap6 is via a driver and wteuiap6 involves other daemons as discussed briefly in the remainder of this chapter. **a** in Figure 166 on page 219 represents the user's entrance into the wteuiapx applications, including wtdriver6/wteuiap6. This similarity in use from

the user's point of view allows for an easy transition from prior wtdriver approaches into the current wtdriver6/wteuiap6 support.

As wteuiap6 was developed, it was decided to define daemons within the NetView for AIX structure, summarized in Figure 167 on page 220. This means that as users of NetView for AIX use commands such as ovstart, the wtxxxxx daemons will be identified. If the user suspects an error with wtxxxxx, the user should contact ITSO via normal non-defect product support; wtxxxxx daemons are *not* part of NetView for AIX The wtxxxxx daemons implement NetView for AIX standard support, but are only user applications and not part of the product code.

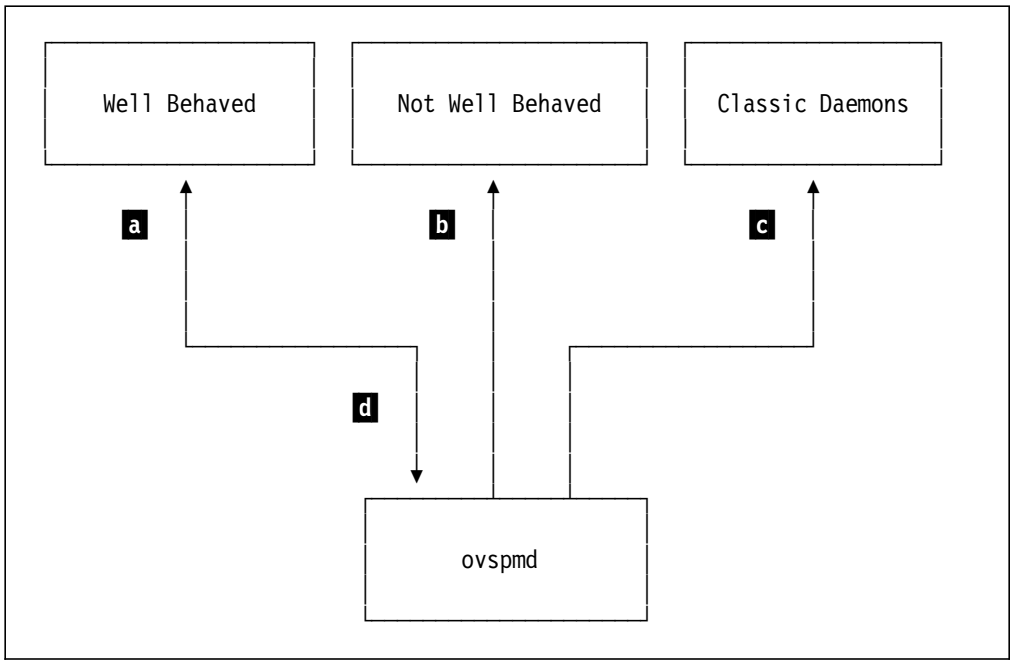


Figure 167. Overall NetView for AIX Daemon Structure

The ovspmd daemon controls all the daemons of NetView for AIX . Daemons are classified by "interaction quality" :

- **a** "Well Behaved" programs

  (OVs_WELL_BEHAVED in .lrf file)
- **b** "Not Well Behaved" programs

  (OVs_NOT_WELL_BEHAVED in .lrf file)
- **c** Classic Daemons

  (OVs_DAEMON in .lrf file)

Well behaved programs are launched by ovspmd and report status to it **d** . ovspmd tracks the process status and last message issued. It also manages an intelligent timeout for startups and shutdowns.

Not well behaved programs are launched by osvpmd but the only thing it can do is check if they are still there (or have died) and kill them on demand (ovstop).

Classic daemons are the common unix daemons; they go into the background on their own initiative and ovspmd loses control of them.

It was decided, as wteuiap6 evolved, to make the application OVs_WELL_BEHAVED. It helped in debugging and program development.

7.2.1 wteuiap6 Addressing the Multiple Operator Requirement

wteuiap6 solution was to add a daemon (wtpbbxd) which maintains synchronization between different operators as shown in Figure 168 on page 221.

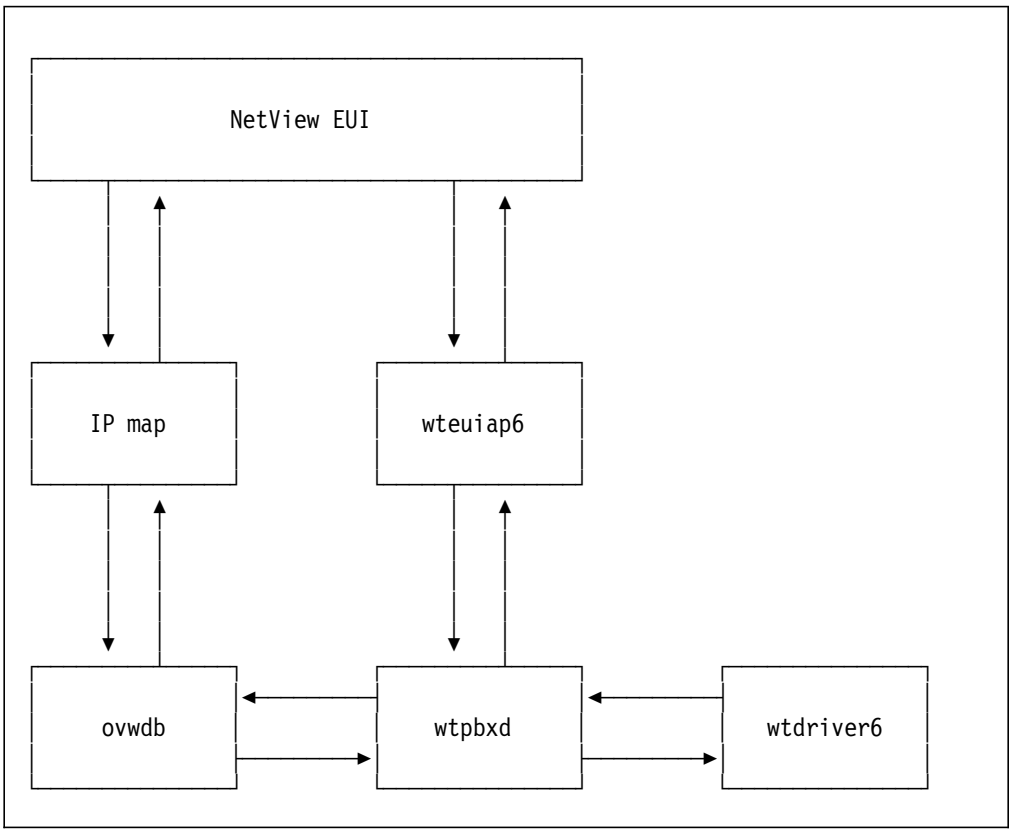


Figure 168. wteuiap6 Daemon Structure

wtpbxd registers for map open/close events and implements wtdriver6 requests on all open maps (the user can specify which one). The default is the Map originally opened by the operator at a particular workstation/process id.

Figure 169 on page 222 summarizes the implemented approach.

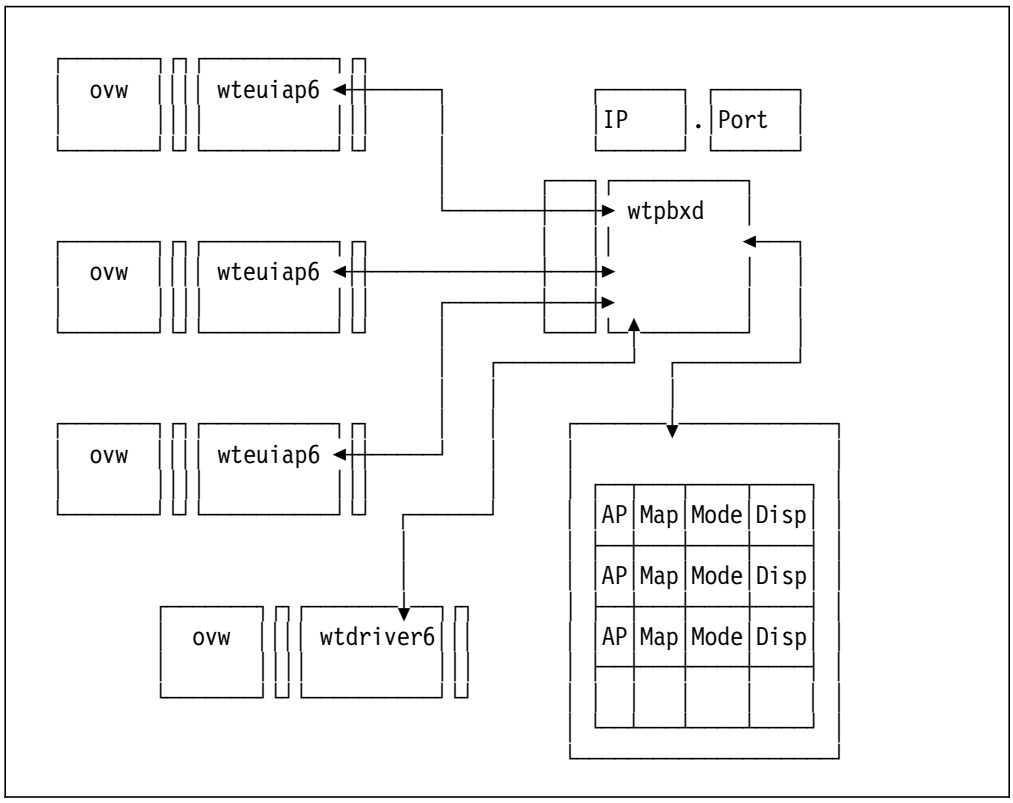


Figure 169. wteuiap6 and Multiple EUIs

7.3 Installing and Managing wteuiap6

The ″make install″ will compile all the modified sources and install them in the correct places.

The following summarizes the install and maintenance functions which are provided:

```
 make install    - To compile and install on the NetView directories

 make uninstall  - To remove wt6 from the NetView directories

 make tar        - To create a tar file with your source

 make floppy     - To create a tar floppy with your source


 make            - To compile wt6.


Files:
wtpbxd.lrf   - Describes wtpbxd to ovspmd (On /usr/OV/lrf)

wteuiap6.reg - Describes wteuiap6 to ovw  (On /usr/OV/registration/C)


Programs:         (On /usr/OV/raleigh - Add it to your $PATH)
wtpbxd       - will be launched by ovspmd (NetView daemon)

wteuiap6     - will be launched by ovw    (NetView End-User Interface)

wtdriver6    - will be run by you on the command line or shell scripts
```

Figure 170. wteuiap6 Install and Maintenance Options

After ″make install″ you (as a user) interface with wtdriver6.

It, in turn, drives wteuiap6.

In the background (you don't care!) is wtpbxd.

After ″make install″ the routines are in /usr/OV/raleigh as:

```
       -rwxr-xr-x-   1 root     system     72019 Jan 13 17:57 wtdriver6

       -rwxr-xr-x-   1 root     system    120373 Jan 13 17:57 wteuiap6

       -rwxr-xr-x-   1 root     system    100518 Jan 13 17:57 wtpbxd
```

The user interfaces with the above through wtdriver6.

wtdriver6 -> talks with -> wteuiap6

The user is not concerned with wtpbxd.

7.4 wtdriver6 Functions

The following figure summarizes use of wtdriver6/wteuiap6.

```
Usage: wtdriver6 [flags] command [...]
Flags:
       [-h wtpbxd-hostname]  - Specify the machine that is running wtpbxd
                               (default is local)
       [-b]                  - Send the request to all displays (broadcast)
       [-d target-display]   - Send the request to a specific display
       [-f submap-name]      - Define the focus (just for this command)
       [-m map-name]         - Send the request to a specific map

Commands:
       stat
       msg      message
       focus    submap_name
       popup    submap_name
       submap   submap_name             [layout] [background-image]
       submapof submap_name object_name [layout] [background-image]
       copy     symbol_name submap_name submap_name
       move     symbol_name submap_name submap_name
       sort     submap_name [keys]
       getlabel object_name
       add      symbol_name symbol_type
                    [x y]
                    [label symbol_label]
                    [submap submap_name]
                    [exec appl_name action_name]
       del      symbol_name
       connect  symbol_name symbol_name
       set      symbol_name status
       assoc    object_name field_name [field_value]
       delobj   object_name
       cloneseg network_name segment_number
```

Figure 171 (Part 1 of 4). Summary of wtdriver6/wteuiap6 Functions

```
Note:
       keys are:
            l  - symbol label
            t  - symbol type
            s  - symbol status
            o  - object status
            c  - compound status

       some symbol types are:  mf     for  "Computer:Main Frame"
                               ws     for  "Computer:Workstation"
                               cr     for  "Connector:Multi-port"
                               ap     for  "Software:License"
                               ss     for  "Software:Process"

       [ If you want to check for the abbreviation table, ]
       [ edit wteuiap6.c and search for symabbrev         ]

                           or anything valid in /usr/OV/symbols/C
              such as from:        /usr/OV/symbols/C/Cards

                          "Cards:Audio"
                          "Cards:Video"
                          "Cards:Thin LAN"

                           another example, from:
                                        /usr/OV/symbols/C/Server
                          "Server:File System"

      set status may be:  unknown
                          normal
                          marginal
                          critical
                          acknowledge
                          up
                          down
```

Figure 171 (Part 2 of 4). Summary of wtdriver6/wteuiap6 Functions

```
5. Some User fields are:

     /*****
     ***
     * Field Registration file for wteuiap6
     * isUP field for application identify the component is UP or Down
     ***
     *****/

     Field "Software Status" {
             Type    StringType;
     }
     Field "mqseries_field_one" {
             Type StringType;
     }
     Field "IDNX_Field_One" {
             Type StringType;
     }
     Field "IDNX_Field_Two" {
             Type StringType;
     }
     Field "WT Merge Id" {
             Type StringType;
     }
     Field "Some Integer" {
             Type    Integer32;
     }
     Field "Some String" {
             Type    StringType;
     }

6. The wteuiap6  registration file (in /usr/OV/registration/C) is:
   (This is used at EUI initiation time)

   /*
    Registration for OVw API sample application WTEUIAP6
    @(#)$Revision: 1.9 $ $Date: 1994/08/30 21:13:31 $
   */

   Application "OVw API Example WTEUIAP6" {

      // wteuiap6 resided in the path /usr/OV/raleigh/wteuiap6
      // it will be initated after NV/6k is running .

      Command -Initial -Shared -Restart "/usr/OV/raleigh/wteuiap6";
   }

7. The wtpbxd registration (in /usr/OV/lrf) is:
   (This is used at ovstart initiation time)
```

Figure 171 (Part 3 of 4). Summary of wtdriver6/wteuiap6 Functions

```
    wtpbxd:/usr/OV/raleigh/wtpbxd:
    OVs_YES_START:ovwdb::OVs_WELL_BEHAVED:10:


  8. When adding an icon with a submap below it ("submap" option in "add"
     function above), the default submap name will be the same as the
     icon it is hung from

  9. When adding an executable icon ("exec" option in "add" function
     above), the "appl_name" and "action_name" come from entries in the
     registration files. For example if you look in /usr/OV/registration/C
     you will find a file "mailtool" containing appl_name "Mail Tool" and
     action_name "mailtool". To create a symbol that executes this you would
     do something like:   wtdriver6 add Mail ss exec "Mail Tool" mailtool
```

Figure 171 (Part 4 of 4). Summary of wtdriver6/wteuiap6 Functions

7.5 Output of wtdriver6 stat

The following is an example of what wtpbxd is managing when multiple NetView for AIX operators are online with the EUI and open maps in read/write mode. The command entered was: wtdriver6 stat.

```
pbxd: Statistics:
      Total Calls: 5
      With a bad magic number: 0
      With a bad version number: 0
      With a bad command number: 0
      Which leaves 5 good calls

EUI Table, 3 entries

Entry Mode   Map                       X-Display            Application
            Pid    IP Address.Port

[  0]  RW    shogm1                   rs60003:0.0       EUI Application 6
             50412 127.0.0.1.3396

[  1]  RW    robmacg                  itsoxst46:0.0     EUI Application 6
             58837 127.0.0.1.4832

[  2]  RW    barry                    itsoxst43:0.0     EUI Application 6
             65486 127.0.0.1.1169
```

In the above situation, commands could (for example) be targeted to the NetView for AIX maps as:

```
wtdriver6 -d rs60003:0.0 -m shogm1  submap inventory
wtdriver6 -d rs60003:0.0 -m shogm1  add    appl1 ap 50 10
wtdriver6 -m rs60003:0.0 -m shogm1  set    appl1 up

wtdriver6 -d itsoxst46:0.0 -m robmacg submap finance
wtdriver6 -d itsoxst46:0.0 -m robmacg add    appl2 ap 50 10
wtdriver6 -d itsoxst46:0.0 -m robmacg set    appl2 up

wtdriver6 -d itsoxst43:0.0 -m barry   submap forums
wtdriver6 -d itsoxst43:0.0 -m barry   add    appl3 ap 50 10
```

```
wtdriver6 -d itsoxst43:0.0 -m barry   set    appl3 up
```

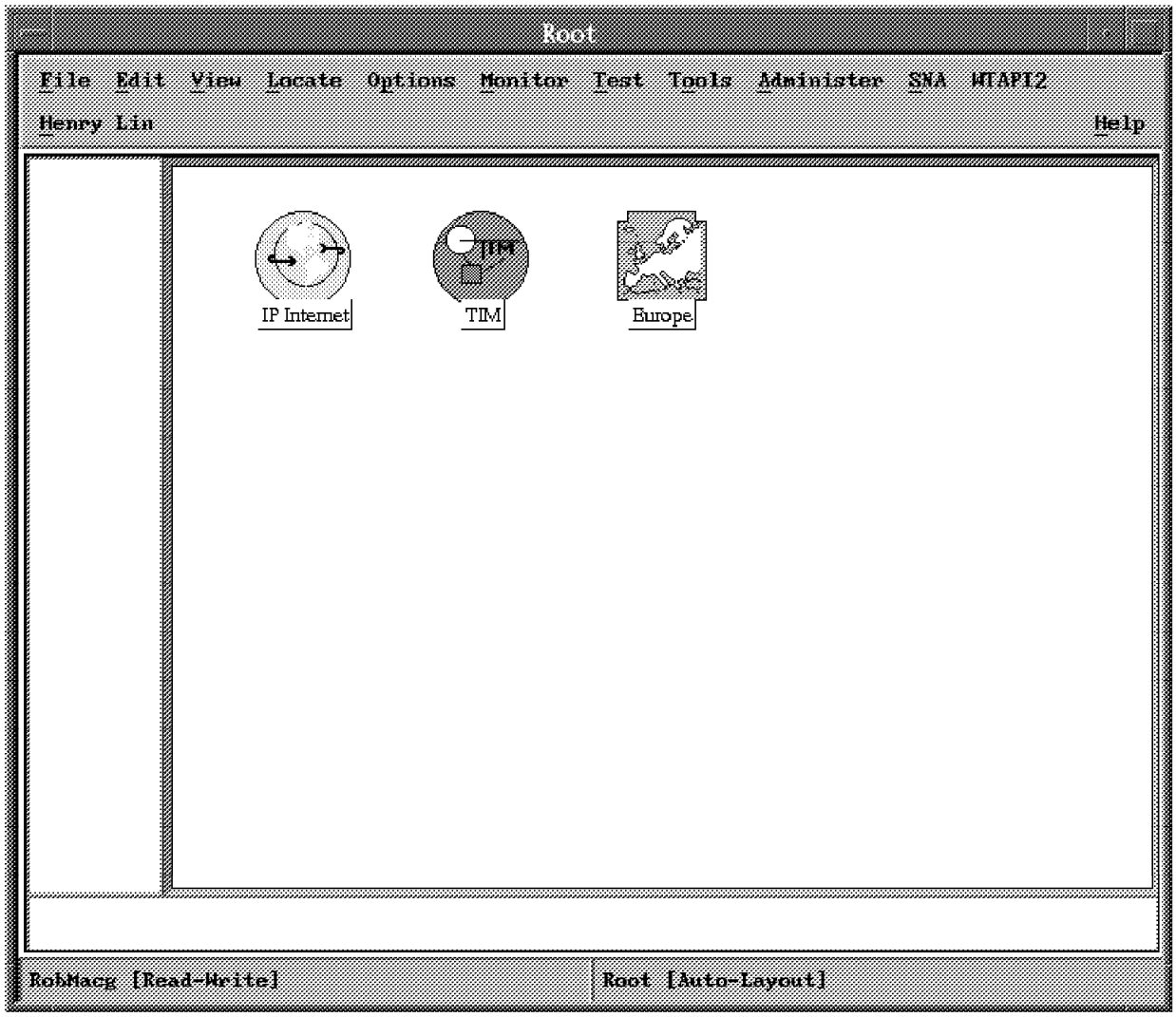
The operator managing shogm1 would see the inventory submaps and resources, the operator managing robmacg would see the finance submaps and resources, and the operator managing barry would see the forums submaps.

All wtdriver6 commands as shown in Figure 171 on page 224 (including copy of submaps for merging resources into one operator's-managed submap) are possible and manageable.

7.6 wteuiap6 Example 1

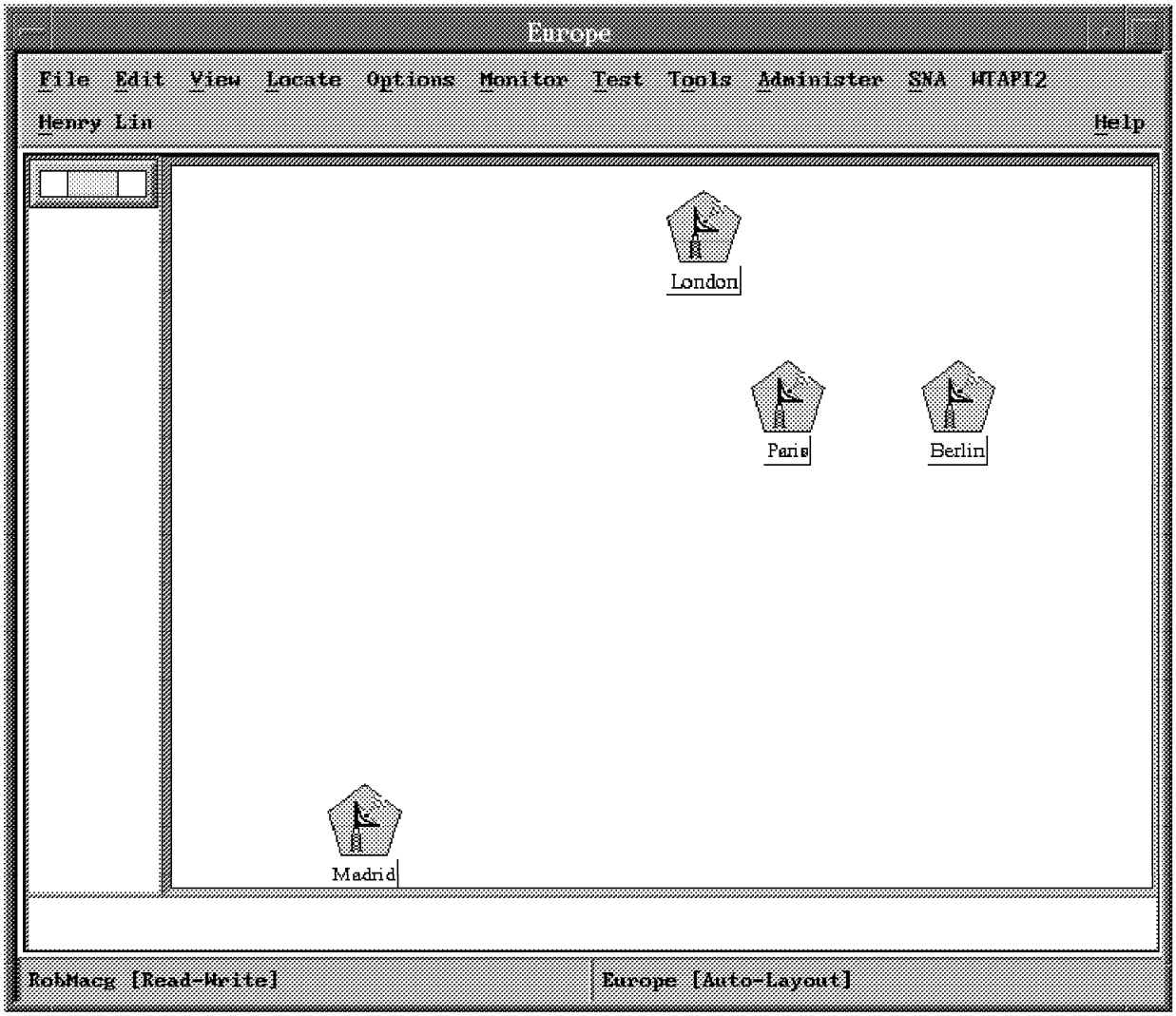
Adding a location symbol with a submap under it to the Root map:

```
wtdriver6 -m RobMacg focus root
wtdriver6 -m RobMacg add Europe Location:.Europe submap
```



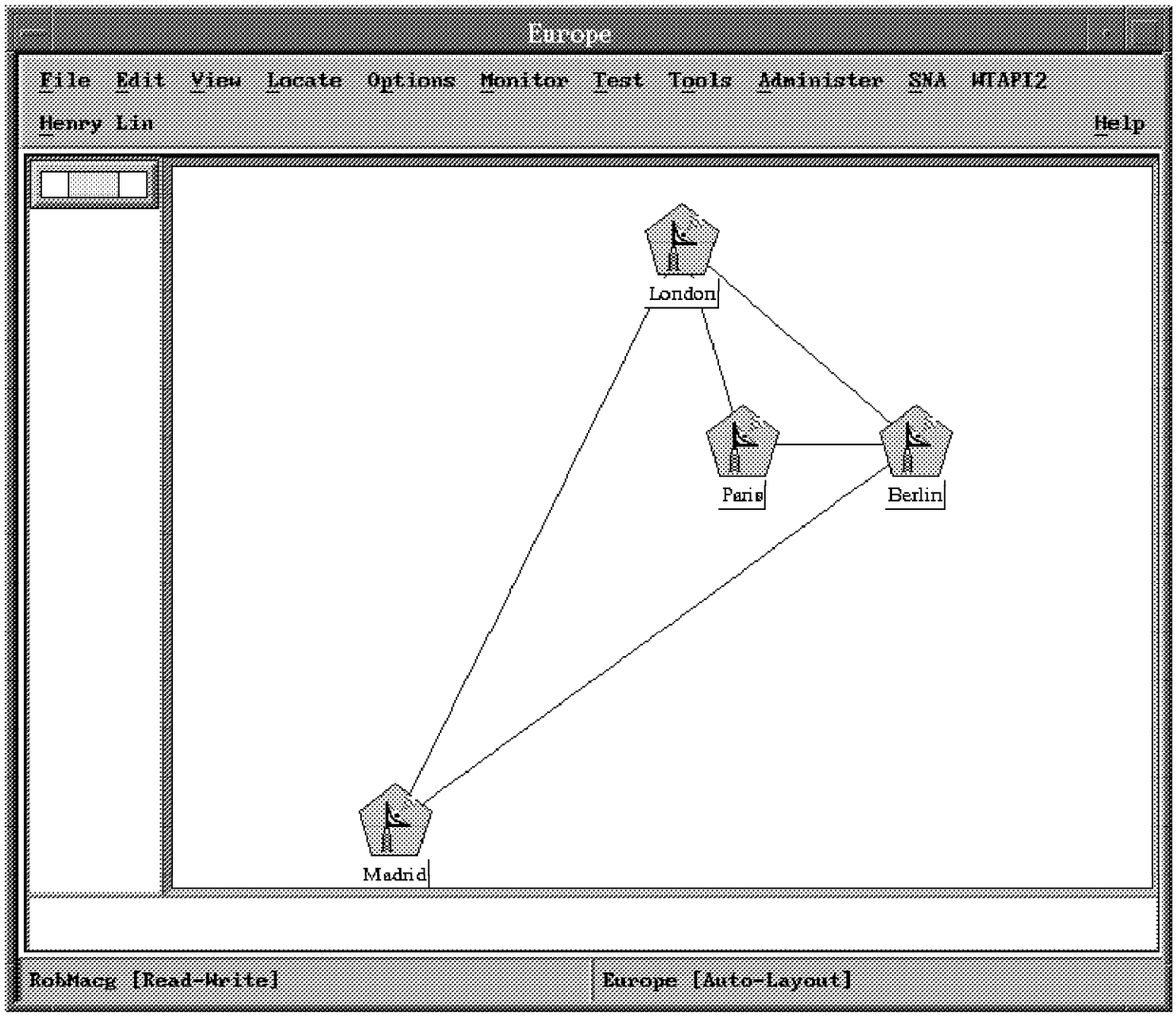
Next, adding some nodes to the new submap:

```
wtdriver6 -m RobMacg focus Europe
wtdriver6 -m RobMacg add London Transceiver:"Satellite equipment" 50 10
wtdriver6 -m RobMacg add Paris  Transceiver:"Satellite equipment" 60 30
wtdriver6 -m RobMacg add Berlin Transceiver:"Satellite equipment" 80 30
wtdriver6 -m RobMacg add Madrid Transceiver:"Satellite equipment" 10 80
```



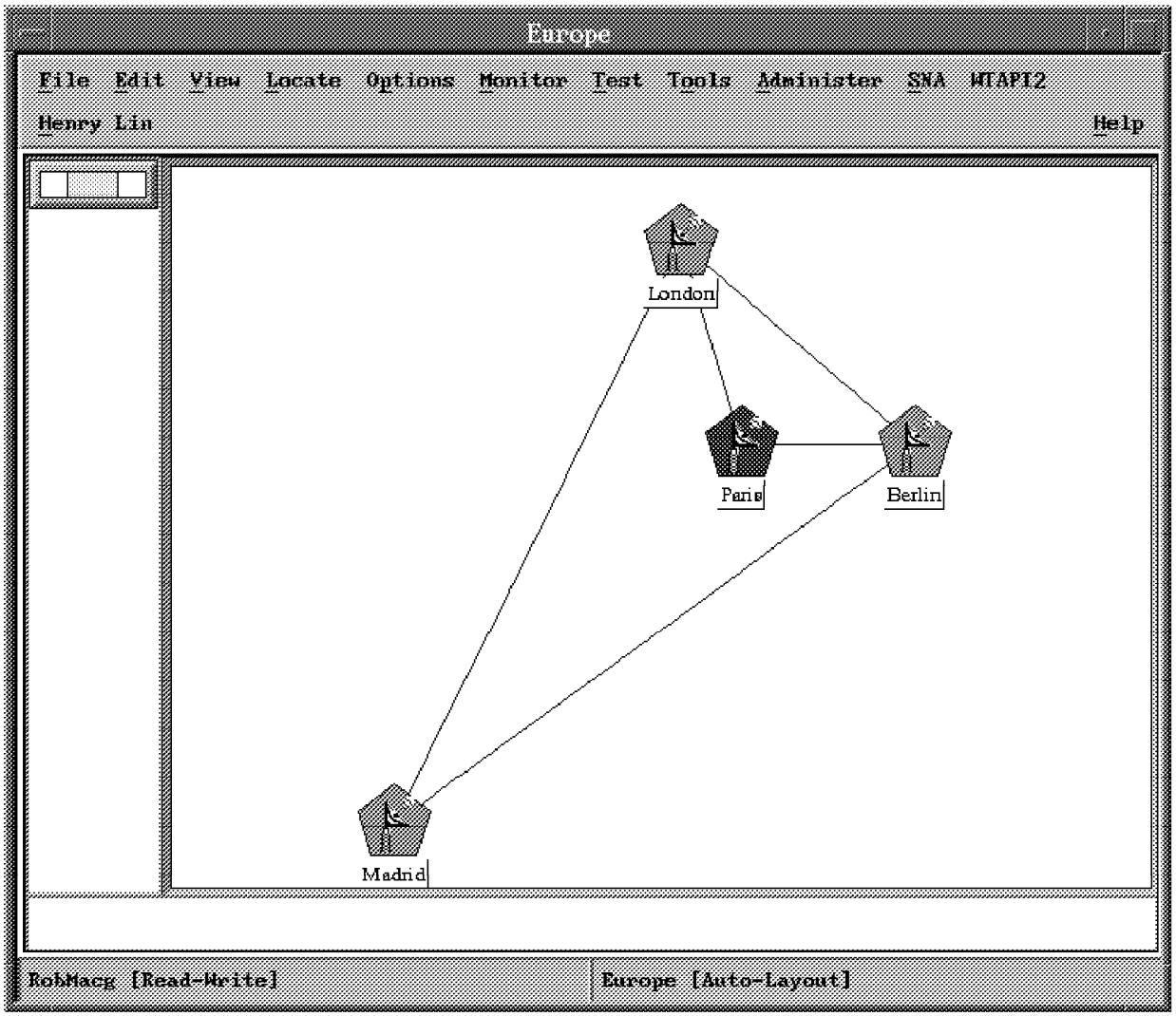
Then, connecting the nodes together (submap focus already done):

```
wtdriver6 -m RobMacg connect London Paris
wtdriver6 -m RobMacg connect London Berlin
wtdriver6 -m RobMacg connect London Madrid
wtdriver6 -m RobMacg connect Berlin Paris
wtdriver6 -m RobMacg connect Berlin Madrid
```



Setting node statuses (Paris is down...):

```
wtdriver6 -m RobMacg set London up
wtdriver6 -m RobMacg set Paris down
wtdriver6 -m RobMacg set Berlin up
wtdriver6 -m RobMacg set Madrid up
```



7.7 wteuiap6 Example 2

The following shell summarizes this example:

```
wtdriver6 -m shogm1 focus Root

wtdriver6 add ITSO_MQSeries Network:Network

wtdriver6 submapof ITSO_MQSeries ITSO_MQSeries

wtdriver6 assoc ITSO_MQSeries "Some String" "If Problems,
                contact: Peter Swanson, Bldg 657, Phone: 01-932-3457"

wtdriver6 -m shogm1 focus ITSO_MQSeries

wtdriver6 add rs60003_mq ap

wtdriver6 add rs60002_mq ap

wtdriver6 assoc rs60003_mq "Some String" "IBM MQSeries.
                Contact Bob Johnson of ITSO-Raleigh"

wtdriver6 assoc rs60002_mq "Some String" "IBM MQSeries.
                Contact Nancy Smith of ITSO-Raleigh"

wtdriver6 connect rs60003_mq rs60002_mq

wtdriver6 connect rs60003_mq rs60002_mq

wtdriver6 -m shogm1 focus "rs60003_mq to rs60002_mq"

wtdriver6 add somequeue ss 50 10 exec  MQSeries_ITSO show_MQSeries_status

wtdriver6 del rs60003_mq

wtdriver6 assoc somequeue "mqseries_field_one" "some queue information"

wtdriver6 set rs60003_mq up

wtdriver6 set rs60002_mq up

wtdriver6 set somequeue up
```

Figure 172. showmq.shell

In addition to the shell, there are two other files involved. For information on their use, refer to NetView for AIX documentation, including *IBM NetView for AIX Programmer's Reference,* SC31-6239.

```
Application "MQSeries_ITSO"
{

    Action "show_MQSeries_status"

    {
        Command "xnmappmon -commandTitle \"MQSeries Queue Information\"
                           -cmd /usr/OV/raleigh/showMQSeries_Q_status

\"$OVwSelection1\"";

        MinSelected 1;
        MaxSelected 1;
    }
}
```

Figure 173. showMQSeries.reg Registration File

```
#!/bin/ksh
#-------------------------------------------------------------#
# Shell to take the name of an IDNX card symbol and display  #
# Object database information about it                       #
#-------------------------------------------------------------#


print "      MQSeries Status Information for $1 "

print "═══════════════════════════════════════════"

print ""

ovobjprint -s $1 | awk '/mqseries | MQSeries/ {print $3, $4, $5}'

print ""

print "═══════════════════════════════════════════"
```

Figure 174. showMQSeries_Q_status

7.8 Execution Panels

The shell (see Figure 172 on page 232) will add a symbol to the Root submap. First, the Root map has only the IP Internet symbol.

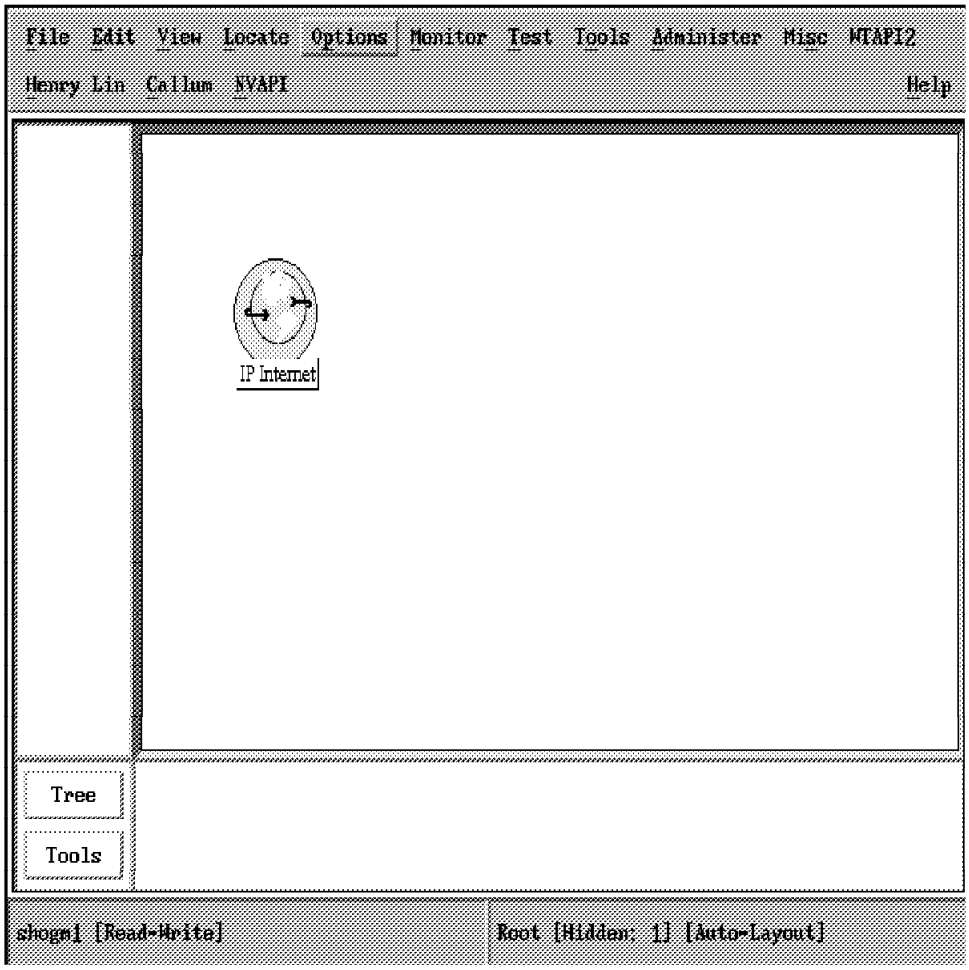


Figure 175. Root Submap Without MQSeries Symbol

Following the shell's execution, a symbol has been added to the Root submap.

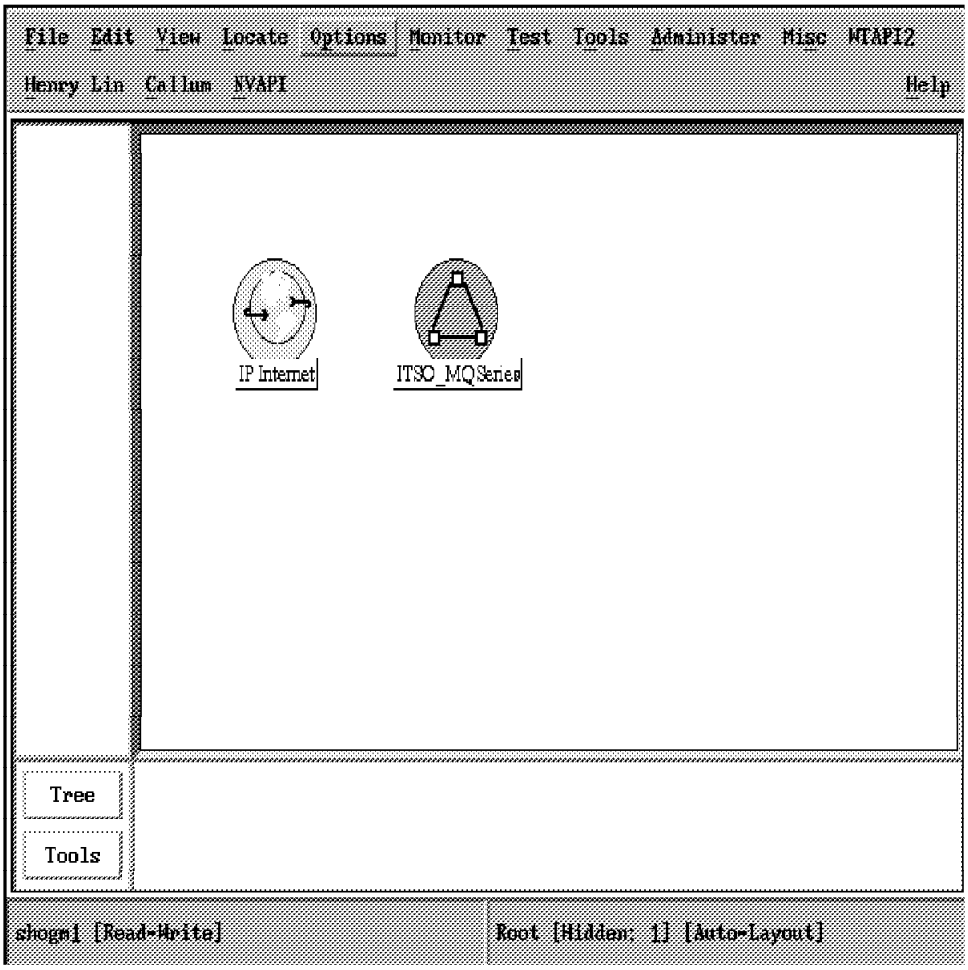


Figure 176. Root Submap After MQSeries Symbol Has Been Added

The added symbol has an associated submap. Clicking on the added symbol **ITSO_MQSeries** will result in the following submap.

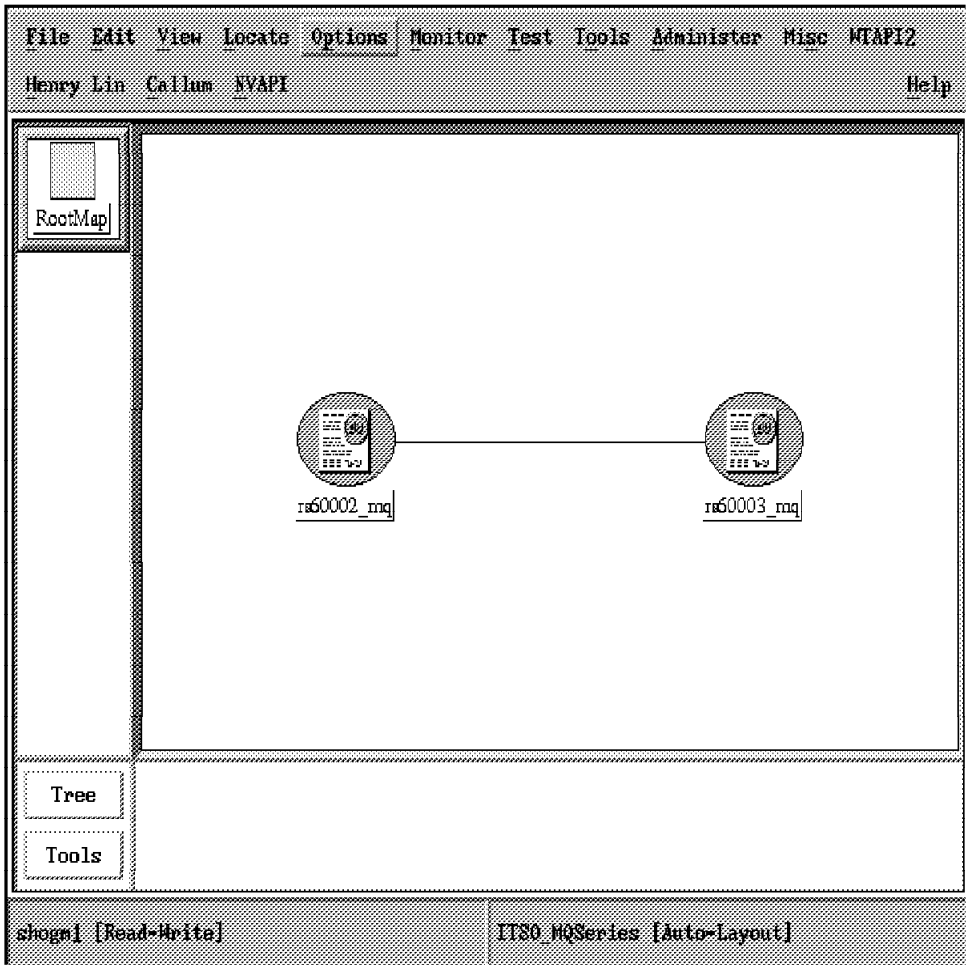


Figure 177. Submap Resulting from Clicking on ITSO_MQSeries

The submap for ITSO_MQSeries has two symbols connected together. See Figure 172 on page 232 for how this was done. The approach used resulted in the connection link being what is known in NetView for AIX terminology as a *Meta-Connection*. Clicking on the connection brings forward a submap which has had an executable symbol added to it by wtdriver6.

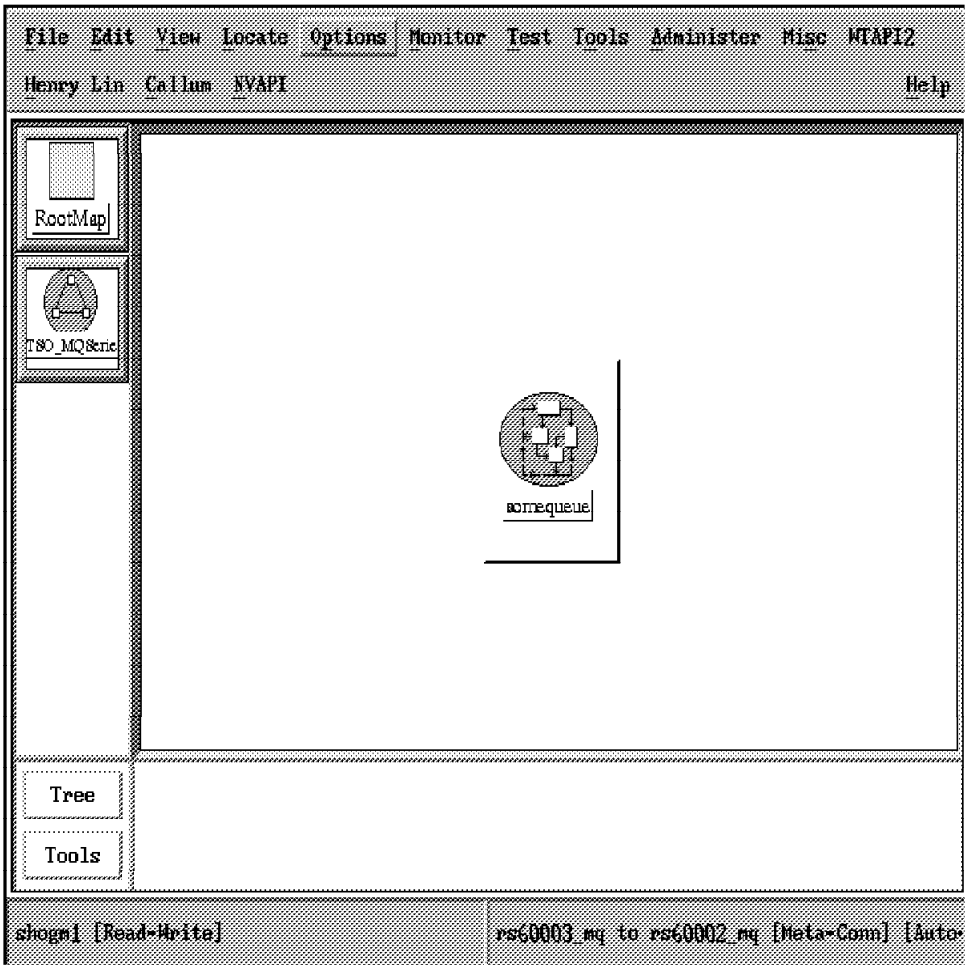


Figure 178. The Meta-Connection Submap

The executable symbol results in a shell being executed (see the registration file in Figure 173 on page 233). The following panel is the result of this shell being driven by the operator clicking on **somequeue**.

The information presented in the resulting panel is incomplete since we have not yet written the code to obtain the information from MQSeries to be presented to the user. That will be done in an upcoming project.

Refer once again to Figure 171 on page 224 for operands that can be used by wtdriver/wteuiap6 to meet a variety of application needs.

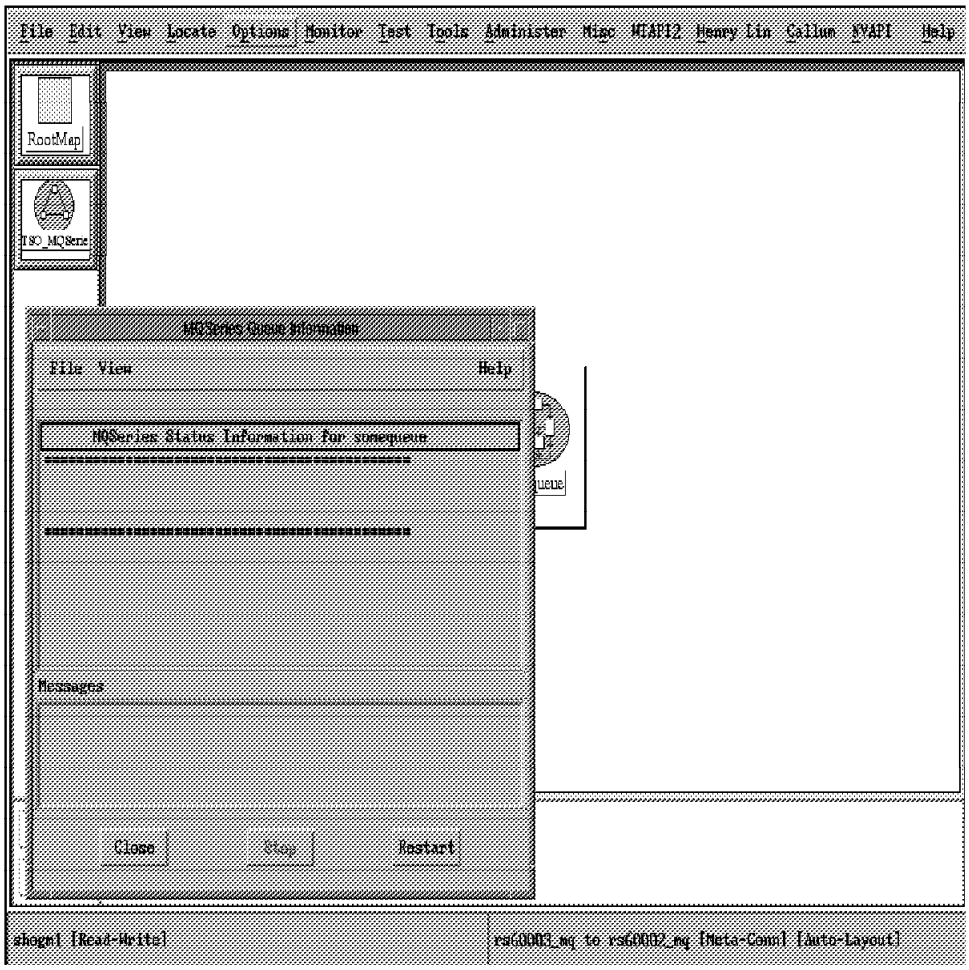


Figure 179. Output of the Shell Driven by Executable Symbol

It might be desirable to have an MQSeries symbol on the same submap as an IP symbol. The following command accomplishes that, as shown in the next figure.

```
wtdriver6 -m shogm1 -f rs60003 add rs60003_mq ap
```

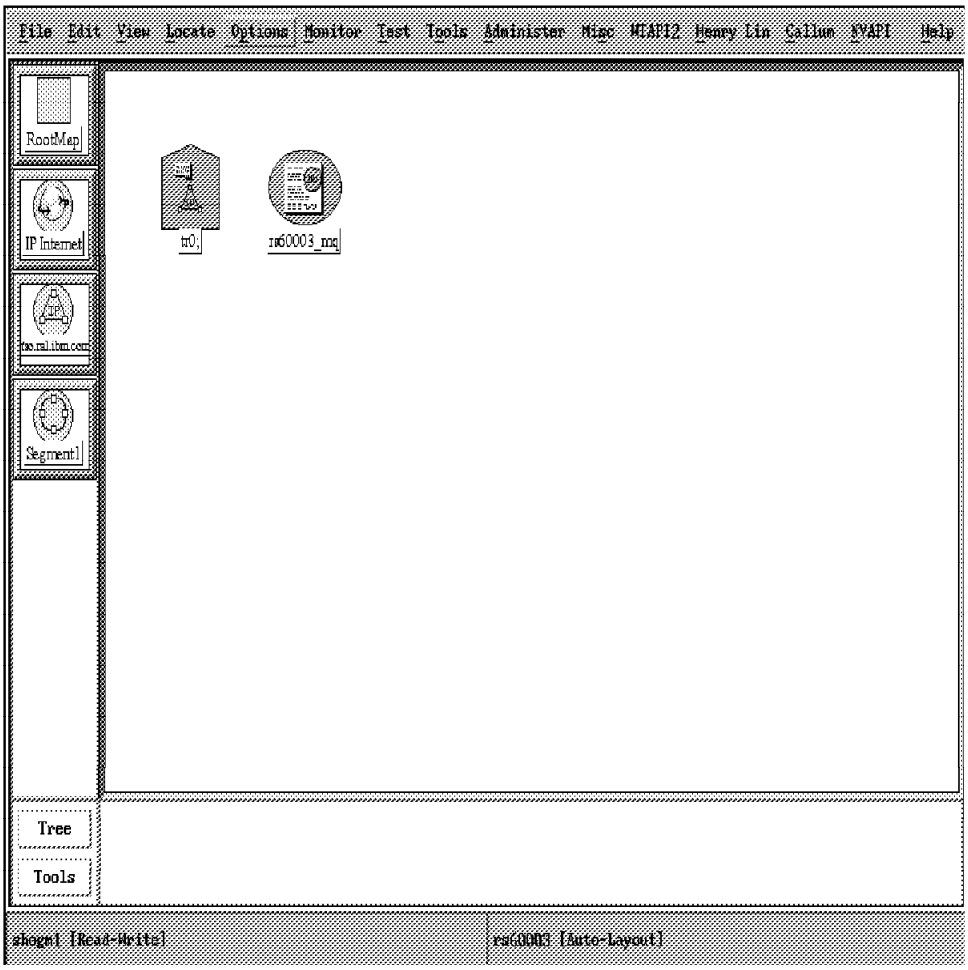


Figure 180. Submap Containing Both IP and MQSeries Symbols

Following execution, looking at an MQSeries object in the NetView for AIX object database, via the command:

```
/usr/OV/bin/ovobjprint -s rs60003_mq
```

results in:

```
                OBJECTID                SELECTION NAME
OBJECT: 1027
        FIELD ID        FIELD NAME                      FIELD VALUE
        10              Selection Name                  "rs60003_mq"
        14              OVW Maps Exists                 1
        15              OVW Maps Managed                1
        66              isSoftware                      TRUE
        129             Software Status                 "Unknown"
        135             Some String                     "IBM MQSeries.
                                        Contact Bob Johnson of ITSO-Raleigh"
```

Having user information in the object database together with information from NetView for AIX and the family of products is headed towards a common repository of network and systems management information.

Stay tuned!

Appendix A. Open Topology MIB Reference

This appendix describes the elements of the Open Topology MIB and the meaning of some of the more important fields.

A.1 The Open Topology MIB

The Open Topology MIB is located in `/usr/OV/snmp_mibs/drafts/ibm-nv6ktopo.mib`.

The data contained in this MIB is stored as tables, and the tables are divided up into groups. Groups contain all tables and variables that describe all the resources of a single class.

The tables defined in the MIB are divided into groups and tables as shown in Table 19.

Table 19. Open Topology Groups and Tables

Group	Tables
Vertex	SAP Vertex
Simple Connection	Simple Connection Underlying Connection
Arc	Arc Underlying Arc
Graph	Graph Members Members Arcs Attached Arcs Additional Members Additional Graphs

The NetView for AIX open topology MIB also defines a set of traps that can be used to communicate topology and status changes to the network management applications. The traps can be used to tell the management applications the following information:

- Newly discovered vertices or connections
- Deletion of network resources
- Changes of resource status
- Changes in variable values

Table 20 on page 242 shows the names of the tables and the trap names that can be used to manipulate them:

Table 20. List of Open Technology Traps	
Table	**Name of the TRAP**
Vertex	newVertex deletedVertex vertexStateChange vertexVariableChange
SAP	newSAPTrap deletedSAP
Simple Connection	newSimpleConnection deletedSimpleConnection simpleConnectionStateChange simpleConnectionVariableStateChange
Underlying Connection	newUnderlyingConnection deletedUnderlyingConnection
Arc	newArc deletedArc arcStateChange arcVariableChange
Underlying Arc	newUnderlyingArc deletedUnderlyingArc
Member Arc	newMemberArc deletedMemberArc
Members	newMemberTrap deletedMember
Additional Member Information	newMemberInformation memberInformationVariableChange
Additional Graph Information	newAdditionalGraphInformation additionalGraphInformationVariableChange

See A.2, “An Open Technology MIB Cross Reference” on page 243 for a description of the contents of each of these tables, and the meaning of some of the Open Topology status fields.

A.2 An Open Technology MIB Cross Reference

A.2.1 Vertex Group: .1.3.6.1.4.1.2.5.3.1

A.2.1.1 Vertex Table: .1.3.6.1.4.1.2.5.3.1.1.1.x

The vertex table is used to represent all communicating entities in a topology. There is one entry for each entity that an agent is aware of.

```
newVertex            1879048192 '70000000'h
deletedVertex        1879048193 '70000001'h
vertexStateChange    1879048194 '70000002'h
vertexVariableChange 1879048195 '70000003'h

.1.3.6.1.4.1.2.5.3.1.1.1.1  integer          vertexType                other(1), invalid(2), vertex(3)
.1.3.6.1.4.1.2.5.3.1.1.1.2  vertexprotocol   vertexProtocol            ** integer(vertex), .1.3.6.1.2.1.2.2.1.3.x(graph)
.1.3.6.1.4.1.2.5.3.1.1.1.3  octetstring      vertexName
.1.3.6.1.4.1.2.5.3.1.1.1.4  integer          vertexLabel
.1.3.6.1.4.1.2.5.3.1.1.1.5  integer          vertexMine
.1.3.6.1.4.1.2.5.3.1.1.1.6  octetstring      vertexLocation
.1.3.6.1.4.1.2.5.3.1.1.1.7  objectidentifier vertexManagementExtension
.1.3.6.1.4.1.2.5.3.1.1.1.8  octetstring      vertexManagementAddr
.1.3.6.1.4.1.2.5.3.1.1.1.9  objectidentifier vertexIcon
.1.3.6.1.4.1.2.5.3.1.1.1.10 integer          vertexOperationnalState
.1.3.6.1.4.1.2.5.3.1.1.1.11 integer          vertexUnknownStatus
.1.3.6.1.4.1.2.5.3.1.1.1.12 integer          vertexAvailabilityStatus
.1.3.6.1.4.1.2.5.3.1.1.1.13 integer          vertexAlarmStatus
```

A.2.1.2 SAP Table: .1.3.6.1.4.1.2.5.3.1.2.1.x

This table shows the mapping of vertices used and provides SAPs.

```
NewSAP               1879048196 '70000004'h
deletedSAP           1879048197 '70000005'h
SAPVariableChange    1879048198 '70000006'h

.1.3.6.1.4.1.2.5.3.1.2.1.1  integer          sapType           valid(1), invalid(2)
.1.3.6.1.4.1.2.5.3.1.2.1.2  vertexProtocol   sapVertexProtocol ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.1.2.1.3  octetstring      sapVertexName
.1.3.6.1.4.1.2.5.3.1.2.1.4  integer          sapIndexId
.1.3.6.1.4.1.2.5.3.1.2.1.5  integer          sapServiceType    using(1) providing(2)
.1.3.6.1.4.1.2.5.3.1.2.1.6  vertexProtocol   sapProtocol       ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.1.2.1.7  octetstring      sapAddress
```

A.2.2 Simple Connection Group: .1.3.6.1.4.1.2.5.3.2

A.2.2.1 Simple Connection Table: .1.3.6.1.4.1.2.5.3.2.1.1.x

This table contains one entry for each connection with an endpoint known about by an agent. It contains information about each connection from the perspective of an endpoint.

```
newSimpleConnection      1879048199 '70000007'h
deletedSimpleConnection  1879048200 '70000008'h
simpleConnStateChange    1879048201 '70000009'h
simpleConnVariableChange 1879048202 '7000000A'h

.1.3.6.1.4.1.2.5.3.2.1.1.1  integer          simpleConnType                other(1), invalid(2), simpleConnection(3)
.1.3.6.1.4.1.2.5.3.2.1.1.3  vertexProtocol   localEndpointProtocol         ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.2.1.1.4  octetstring      localEndpointName
.1.3.6.1.4.1.2.5.3.2.1.1.5  integer          simpleConnIndexId
.1.3.6.1.4.1.2.5.3.2.1.1.6  octetstring      simpleConnName
.1.3.6.1.4.1.2.5.3.2.1.1.8  vertexProtocol   connectionPartnerProtocol     ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.2.1.1.9  integer          connectionPartnerName
.1.3.6.1.4.1.2.5.3.2.1.1.11 octetstring      simpleConnManagementExtension
.1.3.6.1.4.1.2.5.3.2.1.1.12 octetstring      simpleConnManagementAddress
.1.3.6.1.4.1.2.5.3.2.1.1.13 objectidentifier simpleConnManagementIcon
.1.3.6.1.4.1.2.5.3.2.1.1.14 integer          simpleConnOperationnalState
.1.3.6.1.4.1.2.5.3.2.1.1.15 integer          simpleConnUnknownStatus
.1.3.6.1.4.1.2.5.3.2.1.1.16 integer          simpleConnAvailabilityStatus
.1.3.6.1.4.1.2.5.3.2.1.1.17 integer          simpleConnAlarmStatus
```

A.2.2.2 Underlying Connection Table: .1.3.6.1.4.1.2.5.3.2.2.1.x

This table provides the mapping from using simple connection to the underlying connection. There is one entry for each simple connection used by another simple connection.

```
newUnderlyingConnection      1879048203 '7000000B'h
deletedUnderlyingConnection  1879048204 '7000000C'h

.1.3.6.1.4.1.2.5.3.2.2.1.1  integer          ulcType                 valid(1), invalid(2)
.1.3.6.1.4.1.2.5.3.2.2.1.2  vertexProtocol   ulcEndpointProtocol     ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.2.2.1.3  octetstring      ulcEndpointName
.1.3.6.1.4.1.2.5.3.2.2.1.4  integer          ulcEndpointId
.1.3.6.1.4.1.2.5.3.2.2.1.5  integer          ulcIndexId
.1.3.6.1.4.1.2.5.3.2.2.1.6  integer          underlyngConnectionKind
.1.3.6.1.4.1.2.5.3.2.2.1.7  vertexProtocol   uconnEndpointprotocol   ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.2.2.1.8  octetstring      uconnEndpointName
.1.3.6.1.4.1.2.5.3.2.2.1.9  integer          uconnSimpleConnId
.1.3.6.1.4.1.2.5.3.2.2.1.10 integer          nextSerialUlcIndexId
```

A.2.3 Arc Group: .1.3.6.1.4.1.2.5.3.3

A.2.3.1 Arc Table: .1.3.6.1.4.1.2.5.3.3.1.1.x

The arc table contains one entry for every arc represented by an agent. An agent is not required to support this table. The arc table must be supported if the agent supports the graph group.

```
newSimpleArc      1879048206 '7000000E'h
deletedArc        1879048207 '7000000F'h
arcStateChange    1879048208 '70000010'h
arcVariableChange 1879048209 '70000011'h

.1.3.6.1.4.1.2.5.3.3.1.1.1  integer          arcType                other(1), invalid(2), arc(3)
.1.3.6.1.4.1.2.5.3.3.1.1.2  octetstring      arcLabel
.1.3.6.1.4.1.2.5.3.3.1.1.4  vertexProtocol   aEndpointProtocol      ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.3.1.1.5  octetstring      aEndpointName
.1.3.6.1.4.1.2.5.3.3.1.1.7  vertexProtocol   zEndpointProtocol      ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.3.1.1.8  octetstring      aEndpointName
.1.3.6.1.4.1.2.5.3.3.1.1.9  integer          arcIndexId             --> maArcIndexId in Members Arc Table (.9)
.1.3.6.1.4.1.2.5.3.3.1.1.10 integer          aDetailsIndexId
.1.3.6.1.4.1.2.5.3.3.1.1.11 integer          zDetailsIndexId
.1.3.6.1.4.1.2.5.3.3.1.1.12 objectidentifier arcManagementExtension
.1.3.6.1.4.1.2.5.3.3.1.1.13 octetstring      arcManagementAddr
.1.3.6.1.4.1.2.5.3.3.1.1.14 objectidentifier arcIcon
.1.3.6.1.4.1.2.5.3.3.1.1.15 integer          arcOperationnalState
.1.3.6.1.4.1.2.5.3.3.1.1.16 integer          arcUnknownStatus
.1.3.6.1.4.1.2.5.3.3.1.1.17 integer          arcAvailabilityStatus
.1.3.6.1.4.1.2.5.3.3.1.1.18 integer          arcAlarmStatus
```

A.2.3.2 Underlying Arc Table: .1.3.6.1.4.1.2.5.3.3.2.1.x

This table provides the mapping from the using arc to the underlying arc. It also depicts an ordered list of serial underlying arcs.

```
newUnderlyingArc      1879048210 '70000012'h
deletedUnderlyingArc  1879048211 '70000013'h

.1.3.6.1.4.1.2.5.3.3.2.1.1  integer          ulaType                valid(1), invalid(2)
.1.3.6.1.4.1.2.5.3.3.2.1.2  vertexProtocol   ulaAendpointProtocol   ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.3.2.1.3  octetstring      ulaAendpointName
.1.3.6.1.4.1.2.5.3.3.2.1.4  vertexProtocol   ulaZendpointProtocol   ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.3.2.1.5  octetstring      ulaZendpointName
.1.3.6.1.4.1.2.5.3.3.2.1.6  integer          ulaArcIndexId
.1.3.6.1.4.1.2.5.3.3.2.1.7  integer          ulaIndexId
.1.3.6.1.4.1.2.5.3.3.2.1.8  integer          underlyingArckind      serial(1),parrallel(2)
.1.3.6.1.4.1.2.5.3.3.2.1.9  vertexProtocol   uconnAendpointProtocol ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.3.2.1.10 octetstring      uconnAendpointName
.1.3.6.1.4.1.2.5.3.3.2.1.11 vertexProtocol   uconnZendpointProtocol ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.3.2.1.12 octetstring      uconnZendpointName
.1.3.6.1.4.1.2.5.3.3.2.1.13 integer          uconnArcIndexId
.1.3.6.1.4.1.2.5.3.3.2.1.14 integer          nextSerialUlaIndexId
```

A.2.4 Graph Group: .1.3.6.1.4.1.2.5.3.4

A.2.4.1 Graph Table: .1.3.6.1.4.1.2.5.3.4.1.1.x

There is one entry in the graph table for each graph defined by an agent.

```
newGraph            1879048213 '70000015'h
deletedGraph        1879048214 '70000016'h
graphVariableChange 1879048215 '70000017'h

.1.3.6.1.4.1.2.5.3.4.1.1.1  integer          graphType               graph(3),box(4)
.1.3.6.1.4.1.2.5.3.4.1.1.2  objectidentifier graphProtocol
.1.3.6.1.4.1.2.5.3.4.1.1.3  octetstring      graphName
.1.3.6.1.4.1.2.5.3.4.1.1.4  integer          LayoutAlgorithm         none(1), userdefined(2), pointToPoint(3)
                                                                     bus(4), star(5), spoked-ring(6), rowcol(7)
                                                                     pointToPoint-ring(8), tree(9)
.1.3.6.1.4.1.2.5.3.4.1.1.5  integer          userDefinedLayout
.1.3.6.1.4.1.2.5.3.4.1.1.6  octetstring      graphLocation
.1.3.6.1.4.1.2.5.3.4.1.1.7  octetstring      backgroundMap
.1.3.6.1.4.1.2.5.3.4.1.1.8  objectidentifier graphManagementExtension
.1.3.6.1.4.1.2.5.3.4.1.1.9  octetstring      graphManagementAddr
.1.3.6.1.4.1.2.5.3.4.1.1.10 objectidentifier graphIcon
.1.3.6.1.4.1.2.5.3.4.1.1.11 integer          isRoot                  true(1),false(2)
.1.3.6.1.4.1.2.5.3.4.1.1.12 octetstring      graphLabel
```

A.2.4.2 Members Arc Table: .1.3.6.1.4.1.2.5.3.4.2.1.x

This table shows the member arcs of graphs in the graph table.

```
newMemberArc            1879048216 '70000018'h
deletedMemberArc        1879048217 '70000019'h
memberArcVariableChange 1879048218 '7000001A'h

.1.3.6.1.4.1.2.5.3.4.2.1.1  integer          maType              valid(1), invalid(2)
.1.3.6.1.4.1.2.5.3.4.2.1.2  objectidentifier maGraphProtocol
.1.3.6.1.4.1.2.5.3.4.2.1.3  octetstring      maGraphName
.1.3.6.1.4.1.2.5.3.4.2.1.4  integer          maIndexId
.1.3.6.1.4.1.2.5.3.4.2.1.5  vertexProtocol   maAendpointProtocol ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.4.2.1.6  octetstring      maAendpointName
.1.3.6.1.4.1.2.5.3.4.2.1.7  vertexProtocol   maZendpointProtocol ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.4.2.1.8  octetstring      maZendpointName
.1.3.6.1.4.1.2.5.3.4.2.1.9  integer          maArcIndexId        --> arcindexid  in arc table(.9)
```

A.2.4.3 Graph Attached Arcs Table: .1.3.6.1.4.1.2.5.3.4.3.1.x

This table shows the correlation of attached arcs to the graph each is attached to. Each arc in this table has one end point within a graph in the graph table.

```
newGraphAttachedArc             1879048219 '7000001B'h
deletedGraphAttachedArc         1879048220 '7000001C'h
graphAttachedArcVariableChanghe 1879048221 '7000001D'h

.1.3.6.1.4.1.2.5.3.4.3.1.1  integer          aaType              valid(1), invalid(2)
.1.3.6.1.4.1.2.5.3.4.3.1.2  objectidentifier aaGraphProtocol
.1.3.6.1.4.1.2.5.3.4.3.1.3  octetstring      aaGraphName
.1.3.6.1.4.1.2.5.3.4.3.1.4  integer          aaIndexId
.1.3.6.1.4.1.2.5.3.4.3.1.5  vertexProtocol   aaAendpointProtocol ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.4.3.1.6  octetstring      aaAendpointName
.1.3.6.1.4.1.2.5.3.4.3.1.7  vertexProtocol   aaZendpointProtocol ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.4.3.1.8  octetstring      aaZendpointName
.1.3.6.1.4.1.2.5.3.4.3.1.9  integer          aaArcIndexId
```

A.2.4.4 Members Table: .1.3.6.1.4.1.2.5.3.4.4.1.x

This table has one entry for every member of a graph in the graph table.

```
newMemberTrap        1879048222 '7000001E'h
deletedMemberTrap    1879048223 '7000001D'h
memberVariableChange 1879048224 '70000020'h

.1.3.6.1.4.1.2.5.3.4.4.1.1  integer          memberType              valid(1), invalid(2)
.1.3.6.1.4.1.2.5.3.4.4.1.2  objectidentifier memberProtocol
.1.3.6.1.4.1.2.5.3.4.4.1.3  octetstring      memberName
.1.3.6.1.4.1.2.5.3.4.4.1.4  integer          memberIndexId
.1.3.6.1.4.1.2.5.3.4.4.1.6  vertexProtocol   memberComponentProtocol ** vertex(integer), graph(.1.3.6.1.2.1.2.2.1.3.x)
.1.3.6.1.4.1.2.5.3.4.4.1.7  octetstring      memberComponentName
```

A.2.4.5 Additional Members Information Table: .1.3.6.1.4.1.2.5.3.4.5.1.x

This table contains additional information that is specific to a member of graph with no layout algorithm.

```
newMemberInformation            1879048225 '70000021'h
memberInformationVariableChange 1879048226 '70000022'h

.1.3.6.1.4.1.2.5.3.4.5.1.1  integer          amemberType     valid(1), invalid(2)
.1.3.6.1.4.1.2.5.3.4.5.1.2  objectidentifier amemberProtocol
.1.3.6.1.4.1.2.5.3.4.5.1.3  octetstring      amemberName
.1.3.6.1.4.1.2.5.3.4.5.1.4  integer          amemberIndexId
.1.3.6.1.4.1.2.5.3.4.5.1.5  integer          xCoordinate
.1.3.6.1.4.1.2.5.3.4.5.1.6  integer          yCoordinate
```

A.2.4.6 Additional Graph Information Table: .1.3.6.1.4.1.2.5.3.4.6.1.x

This table contains additional information about a graph. It contains entries only for graphs with additional information. The table should include only one row with a non-zero value of graphRootIndexId, the first row with a valid graphrootIndexId will be the hub or center vertex of the graph.

```
newAdditionalGraphInformation           1879048227 '70000023'h
additionalgraphInformationVariableChange 1879048228 '70000024'h

.1.3.6.1.4.1.2.5.3.4.6.1.1  integer          addGraphType     valid(1), invalid(2)
.1.3.6.1.4.1.2.5.3.4.6.1.2  objectidentifier addGraphProtocol
.1.3.6.1.4.1.2.5.3.4.6.1.3  octetstring      addGraphName
.1.3.6.1.4.1.2.5.3.4.6.1.4  integer          addGraphIndexId
.1.3.6.1.4.1.2.5.3.4.6.1.5  integer          graphRootIndexId
.1.3.6.1.4.1.2.5.3.4.6.1.6  octetstring      graphDesc1
.1.3.6.1.4.1.2.5.3.4.6.1.7  integer          graphDescrX
.1.3.6.1.4.1.2.5.3.4.6.1.8  integer          graphDescrY
```

A.3 State Information

State and status information is defined by the ISO 10164-2, State management. This information is summarized below.

A.3.1 Operational State

The operational state can be:

Enabled. The device is available for use

Disabled. The device is unavailable for use.

A.3.2 Status Information

In addition to the operational state, we have three status fields available for use.

Unknown status The state of the resource is unknown, as the agent, is not able to advise us of its state.

Availability status This could be one of the following:

- In test
- Failed
- Power off
- Off line
- Off duty
- Dependency
- Degraded
- Not installed

Alarm status This provides further information about alarms issued for the resource, and the actions taken to resolve them.

- Under repair
- Critical
- Major
- Minor
- Alarm outstanding

A.3.3 Mapping States and Status to NetView for AIX Displays

Table 21 shows the relationship between the states of the objects and the way that NetView for AIX displays this information to the user.

Table 21. Resource to NetView for AIX Status Mapping Table

Operational Status	Unknown Status	Availability Status	Alarm Status	NetView/6000 Status
any	true	any	critical	**critical**
any	true	any	other than critical	**unknown**
enabled	false	empty set	any	**normal**
any	false	off duty	any	**normal**
disabled	false	not installed	any	**unmanaged**
disabled	false	off line	any	**marginal**
disabled	false	dependency	any	**critical**
enabled	false	degraded	critical	**critical**
enabled	false	degraded	other than critical	**marginal**
disabled	false	failed	any	**critical**
disabled	false	power off	any	**critical**
enabled	false	in test	any	**marginal**

So, to get the status change information to the NetView for AIX operator, we have to consider four MIB variables, in each of the objects for which gtmd holds status. Table 22 shows the possible numeric values for these fields. Notice that some of these fields can have more than one status bit set, in which case you have to sum the integer values.

Table 22. Status Field Integer Values

Operational Status	Unknown Status	Availability Status	Alarm Status	Integer Value
disabled	true	inTest	underRepair	**1**
enabled	false	failed	critical	**2**
		powerOff	major	**4**
		offLine	minor	**8**
		offDuty	alarmOutstanding	**16**
		dependency		**32**
		degraded		**64**
		notinstalled		**128**

We can now correlate the information in Table 21 with the information in Table 22 to find out which field values will have to be set in order to affect the status of the displayed NetView for AIX resource.

Status information can be maintained in the following tables:

- Vertex
- Simple connection
- Arc

Appendix B. Automatic Seed File Example Programs

This appendix contains sample automatic seed file programs.

```
Application "Backup" {

    Description {
          "Configuration and Control",
          "For Automated Backup"
    }
    Version "V3R1";
        Copyright {
          "Licensed Program Product:",
          "  NetView for AIX",
          "(C) COPYRIGHT International Business Machines Corp. 1994",
          "    All Rights Reserved",
          "US Government Users Restricted Rights - Use, duplication,",
          "or disclosure restricted by GSA ADP Schedule Contract with",
          "IBM Corp. and its licensors.",
          ""
        }
        HelpDirectory "Distman";
        Command -Shared -Initial -Restart "${BackupDir:-/usr/OV/bin}/backup";

        MenuBar "Administer"  {
                "Backup"       _B    f.menu     "backup";
        }
        Menu "backup" {
              "Backup Configuration..."  _B    f.action   "backupConfig";
              "Read Seedfile"            _R    f.action   "backupReadSeed";
              "Build Seedfile"           _R    f.action   "backupBuildSeed";
        }
    Action "backupConfig" {
        }
    Action "backupReadSeed" {
        Command "${aixterm:-/usr/bin/X11/aixterm} -fg black -bg grey
                -T \"Load\ Seed\ File\" -e /u/paul/progs/req_seed.ksh
                >/dev/null 2>&1";
    }
    Action "backupBuildSeed" {
        MinSelected 0;
        Command "${aixterm:-/usr/bin/X11/aixterm} -fg black -bg grey
                -T \"Build\ Seed\ File\" -e /u/paul/progs/build_seed.ksh
                >/dev/null 2>&1";
        }
}
```

Figure 181. Backup Registration File (/usr/OV/registration/C/backup)

```
#!/bin/ksh
# Script to build the seed file from NetView/60000
# Input from keyboard - Manager Node and Backup Node
# The contents of the seed file are built from the NetView/6000 Map
SEED=/u/paul/seed/seed_file

echo "Do you wish to (c)reate or (a)ppend to the Seedfile: \c"; read ACT
if [ "$ACT" = "q" ]
then
    exit 0
else
    if [ -z "$ACT" ]
    then
         echo "Re-enter value"
         sleep 1; exec $0
else
    if [ "$ACT" != "c" ] &&  [ "$ACT" != "a" ]
    then
         echo "Re-enter value"
         sleep 1; exec $0
         fi
    fi
fi

echo "Enter Nodename for Manager (q) to Quit:\c "; read MAN
if [ "$MAN" = "q" ]
then
    exit 0
else
    if [ -z "$MAN" ]
    then
         echo "Re-enter value"
         sleep 1; exec $0
fi
fi
echo "Enter Nodename for Backup Manager (q) to Quit:\c "; read BAK
if [ "$BAK" = "q" ]
then
    exit 0
else
    if [ -z "$BAK" ]
    then
         echo "Re-enter value"
         sleep 1; exec $0
fi
fi
if [ "$ACT" = "c" ]
then
    /u/paul/progs/build_seedfile $MAN $BAK > $SEED
else
    /u/paul/progs/build_seedfile $MAN $BAK >> $SEED
fi
echo "Seed file: $SEED has been updated; press <RETURN> to continue\c "; read
exit 0
```

Figure 182. build_seed.ksh

```
#!/usr/bin/ksh
echo -n "Enter seedfile or (q) to quit : "
read fnseed
if  [ "$fnseed" = "q" ]
then
    exit 0
fi

if  [ -a  "$fnseed"  ]
then
   echo -n "reading seedfile...$fnseed"
   /usr/OV/bin/backup -s  $fnseed &
   echo -n "Seed File process completed. Check output on screen"
   echo -n "Press <RETURN> to continue \c"; read
   exit 0
else
   echo "File $fnseed does NOT exists. Press <RETURN> to re-enter: \c"; read
fi

exec $0
```

Figure 183. req_seed.ksh

```
/*
 * Program to create and append the seed file
 * for Management Takeover.
 *
 * Author Paul Fearn
 * 24/08/94
 *
 * To Compile:
 * cc -o build_seedfile build_seedfile.c -lovw -lov -lntl
 */

#include <stdio.h>
#include <stdlib.h>

#include <OV/ovw.h>
#include "ovw_print.h"

void usage();
void printMap(char *man);

main(int argc, char *argv[])
{
    extern int optind, opterr;
    extern char *optarg;
    int c;
    OVwBoolean errflag = FALSE;
    OVwBoolean verbose = FALSE;
    OVwBoolean printfields = FALSE;

    int i, rc;
    char *label;
    char *SelectName;
    OVwMapInfo *map;
    OVwObjectIdList *op;
    OVwObjectId *lp;

    if (OVwInit() < 0) {
        fprintf(stderr, "%s\n", OVwErrorMsg(OVwError()));
        exit(1);
    }

    map = OVwGetMapInfo();
    op = OVwGetSelections( map, NULL);

    for ( i=0, lp = op->object_ids; i<op->count; i++, lp++ )
    {
         SelectName=OVwDbObjectIdToSelectionName(*lp);
         printf("\"%s\" \"%s\" \"%s\"\n", argv[1], SelectName, argv[2]);
    }

    OVwDbFreeObjectIdList(op);
    OVwDone();
}
```

Figure 184. 'C' program build_seedfile.c

Appendix C. Open Topology Program Samples

This appendix contains the ″C″ language and shell script samples for driving the Open Topology functions of NetView for AIX that were developed during the project. The programs are:

wtotapi1 This is a C program that uses the NetView for AIX Open Topology API. It may be used to add network objects, connect them together, set status and create SAPs.

wtgtm This is a shell script that generates SNMP traps which conform to the Open Topology MIB. It may be used to add and delete network objects, connect them together, set status and create SAPs. Because it uses the trap interface, wtgtm is compatible with AIX NetView/6000 V2R1 as well as NetView for AIX.

C.1 Program Listing for wtotapi1.c

```
/*********************************************************************/
/*********************************************************************/
/* Sample Program to drive the NV6000 V3 gtm API                     */
/*                                                                   */
/* AUTHOR Rob Macgregor                                              */
/*********************************************************************/
/*********************************************************************/

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <wordexp.h>
#include <nvot.h>
#include <OV/ovw_obj.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                   */
} gtm_cmd_id;
```

Figure 185 (Part 1 of 9). wtotapi1.c Program Listing

```
typedef enum usage_code {
  GENERAL_USAGE,        /* General usage text        */
  SET_PROT_ERR,         /* Invalid set protocol cmd  */
  PROT_NOT_SET,         /* Protocol not yet set      */
  ADD_GENERAL,          /* Help on the ADD command   */
  ADD_GRAPH_FORMAT,     /* Invalid submap format     */
  ADD_VERTEX_FORMAT,    /* Invalid add vertex command*/
  ADD_ARC_FORMAT,       /* Invalid add arc command   */
  ADD_SAP_FORMAT,       /* Invalid add sap command   */
  SET_VERTX_FORMAT      /* Invalid set status command*/
 } usage_code;

/* Global variables - current protocol, current graph name and graph protcol */

char                        *oid_prefix = "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 ;

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} ;


#include "wtothelp.c"     /* "usage()" procedure */

/* 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")) == 0)
   {
    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(strcmp(sym_name, icon) == 0 )
        {
          strcpy(oid, dotted_decimal_oid) ;
          return 0 ;
        }
      }
    }
   }
 printf("No entry for %s found in %s. Try again\n", icon, oid_to_sym_file) ;
 return -1 ;
}
```

Figure 185 (Part 2 of 9). wtotapi1.c Program Listing

```
/* Return number of words in input string */
int WORDS(char *cmd_string)
{
 wordexp_t  string_structure ;
 wordexp(cmd_string, &string_structure, 0) ;
 return string_structure.we_wordc ;
}

/* Return nth word in input string */
char * WORD(char *cmd_string, int wordpos)
{
 wordexp_t  string_structure ;
 wordexp(cmd_string, &string_structure, 0) ;
 return string_structure.we_wordv[wordpos - 1] ;
}

/* Routine to qualify what command was issued */
int cmd_to_table(char *cmd)
{
 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 },
      { 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("wtotapi1: Invalid status, options are: up, down, marginal, unknown\n");
 return(EOF) ;
}

/* Set the protocol number we are currently working with */
void set_cur_protocol(char * args)
{
 if( (current_prot_nbr = atoi(args)) == NULL)
   {
     usage(SET_PROT_ERR) ;
     return ;
   }
```

Figure 185 (Part 3 of 9). wtotapi1.c Program Listing

```
 strcpy(current_prot_oid,oid_prefix) ;
 strcat(current_prot_oid, args) ;
 printf("Current protocol now set to %d\n", current_prot_nbr) ;
 printf("The OID for this protocol is %s\n", current_prot_oid) ;
 return ;
}
/* Set the graph. All "add"s subsequently will be members of the
   graph defined in here */
void set_cur_graph(char *graph_name)
{
  /* We must have set a value for current protocol */
  if (current_prot_nbr == 0)
  {
     usage(PROT_NOT_SET) ;
     return  ;
  }
  /* 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 ;
              }
        }

  strcpy(current_graph, graph_name) ;
  strcpy(current_graph_prot, current_prot_oid) ;
  printf("Objects added from now on will be members of %s\n", current_graph) ;
  printf("Protocol OID is: %s\n", current_graph_prot) ;
  return ;
}

/* 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  */
void add_root_graph(char *args)
{
 nvotReturnCode  retc ;
 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, WORD(args, 1)) ;
 strcpy(graph_icon, WORD(args, 2)) ;
 strcpy(graph_layout_c, WORD(args, 3)) ;
 printf("About to add graph %s to Root submap\n", graph_name) ;
 if (sym_to_oid( graph_icon, &icon_oid) != 0)
    return ;
```

Figure 185 (Part 4 of 9). wtotapi1.c Program Listing

```
 graph_layout = char_to_layout(graph_layout_c) ;
 objid = nvotCreateRootGraph(
            current_prot_oid,
            &graph_name,
            graph_layout,
            "",
            &icon_oid,
            &graph_name,
            &graph_details) ;
 printf("wtotapi1: %s\n", nvotGetErrorMsg(nvotGetError())) ;
 return ;
}
/* Add graph graph  */
void add_graph_graph(args)
{
 nvotReturnCode  retc ;
 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, WORD(args, 1)) ;
 strcpy(graph_icon, WORD(args, 2)) ;
 strcpy(graph_layout_c, WORD(args, 3)) ;
 printf("About to add graph %s to submap %s\n", graph_name, current_graph) ;
 if (sym_to_oid( graph_icon, &icon_oid) != 0)
    return ;
 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("wtotapi1: %s\n", nvotGetErrorMsg(nvotGetError())) ;
 return ;
}

/* Add box graph  */
void add_box_graph(args)
{
 nvotReturnCode  retc ;
 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, WORD(args, 1)) ;
 strcpy(graph_icon, WORD(args, 2)) ;
 strcpy(graph_layout_c, WORD(args, 3)) ;
 printf("About to add boxgraph %s to submap %s\n", graph_name, current_graph) ;
 if (sym_to_oid( graph_icon, &icon_oid) != 0)
    return ;
 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("wtotapi1: %s\n", nvotGetErrorMsg(nvotGetError())) ;
 return ;
}
```

Figure 185 (Part 5 of 9). wtotapi1.c Program Listing

```
/* Add vertex     */
void add_vertex(args)
{
 nvotReturnCode  retc ;
 char   icon_oid[40] ;
 char   vertex_name[MAX_NAME_LEN]  ;
 char   vertex_icon[30] ;
 nvotOctetString vertex_details ;
 OVwObjectId objid ;
 strcpy(vertex_name, WORD(args, 1)) ;
 strcpy(vertex_icon, WORD(args, 2)) ;
 printf("About to add vertex %s to graph %s\n", vertex_name, current_graph) ;
 if (sym_to_oid( vertex_icon, &icon_oid) != 0)
    return ;
 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("wtotapi1: %s\n", nvotGetErrorMsg(nvotGetError())) ;
 return ;
}

/* Add an arc     */
void add_arc(char * args)
{
 nvotReturnCode  retc ;
 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, WORD(args, 1)) ;
 strcpy(&end_pt_z, WORD(args, 2)) ;

/* 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 ;
                      }
```

Figure 185 (Part 6 of 9). wtotapi1.c Program Listing

```
  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) ;
     printf("wtotapi1: %s\n", nvotGetErrorMsg(nvotGetError())) ;
   }

 else printf("Arcs can only be added to a Graph graph, %s is a Box graph\n",
             current_graph) ;
 return ;
}
/* Add a providing or using SAP */
void add_sap(char * args)
{

 nvotReturnCode  retc ;
 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, WORD(args, 1)) ;
if (sap_type[0] == 'p')          /* a "Providing" SAP */
   {
      strcpy(&sap_provider_name, WORD(args, 2)) ;
      printf("About to add providing SAP entry %s, protocol %d\n",
                          sap_provider_name, current_prot_nbr) ;
      retc = nvotCreateProvidingSap(
                  current_prot_nbr,
                  &sap_provider_name,
                  77,
                  &sap_provider_name) ;  /* Note: uses vertex name as SAP name*/
      printf("wtotapi1: %s\n", nvotGetErrorMsg(nvotGetError())) ;
      return ;
    }
if (sap_type[0] == 'u')          /* a "Using" SAP */
    {
      strcpy(&sap_user_name,     WORD(args, 2)) ;
      strcpy(&sap_provider_name, WORD(args, 3)) ;
      sap_provider_prot = atoi(WORD(args, 4)) ;
      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) ;
      retc = nvotCreateUsingSap(
                  current_prot_nbr,
                  &sap_user_name,
                  77,
                  &sap_provider_name) ;
      printf("wtotapi1: %s\n", nvotGetErrorMsg(nvotGetError())) ;
      return ;
    }
  usage(ADD_SAP_FORMAT) ;
}


/* "add" requested - sub-selection  */
void add_sub_select(char * addstuff)
{
char  add_cmd[8] ;
char  args[80] ;
int   wordcount ;
 sscanf(addstuff, "%s %[^\n]", &add_cmd, &args)  ;
 wordcount = WORDS(&args) ;
```

Figure 185 (Part 7 of 9). wtotapi1.c Program Listing

```
 switch (cmd_to_table(&add_cmd))
   {
    case GTM_ADD_BOX   :
      {
       if (wordcount != 3) usage(ADD_GRAPH_FORMAT) ;
       else
        {
         if (strcmp(current_graph, "Root") == 0)
            printf("You cannot add a box graph to the Root submap") ;
         else add_box_graph(&args) ;
        }
      break ;
      }
    case GTM_ADD_GRAPH   :
      {
       if (wordcount != 3) usage(ADD_GRAPH_FORMAT) ;
       else
        {
         if (strcmp(current_graph, "Root") == 0) add_root_graph(&args) ;
         else add_graph_graph(&args) ;
        }
      break ;
      }
    case GTM_ADD_VERTEX   :
      {
       if (wordcount != 2) usage(ADD_VERTEX_FORMAT) ;
       else
        {
         if (strcmp(current_graph, "Root") == 0)
         printf("You must set the parent graph for vertex, use parent command\n");
         else add_vertex(&args) ;
        }
      break ;
      }
     case GTM_ADD_ARC  :
      {
       if (wordcount != 2 ) usage(ADD_ARC_FORMAT) ;
       else add_arc(&args) ;
       break ;
      }
     case GTM_ADD_SAP  :
      {
       if ((wordcount != 2) && (wordcount != 4)) usage(ADD_SAP_FORMAT) ;
       else add_sap(&args) ;
       break ;
      }
     default : usage(ADD_GENERAL) ;
    }
return ;
}
/* Set the status of a vertex - up, down, marginal, unknown */
void set_vertex(char * args)
{
 int vert_status ;
 char   vertex_name[MAX_NAME_LEN] ;
 char   status_requested[8] ;

  if (WORDS(args) != 2)
       {
         usage(SET_VERTX_FORMAT) ;
         return ;
       }
  sscanf(args, "%s %s", &vertex_name, &status_requested) ;
  if ( (vert_status = char_to_status(status_requested)) == EOF) return ;
printf ("status - %d\n", vert_status)  ;
  /* Just issue the request */

 nvotChangeVertexStatus( current_prot_nbr, &vertex_name, vert_status) ;
 printf("wtotapi1: %s\n", nvotGetErrorMsg(nvotGetError())) ;
 return ;
}
```

Figure 185 (Part 8 of 9). wtotapi1.c Program Listing

```
int main(int argc, char **argv)
{

 char       cmd[8] ;
 char   args[80] ;
 int    quit = FALSE ;
 char   hostname[48] ;
 nvotReturnCode retc ;
 static unsigned int ovwdbTimeout = 5;

 strcpy(current_graph, "Root") ;
  if ( argc > 2 )
    {
      usage(GENERAL_USAGE) ;
      exit(-1) ;
    }
  if(argc == 1) gethostname(hostname, 48) ;        /* talk to our own host */
  else strcpy(hostname, argv[1]) ;

 /* 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) ;
     }
 printf("Open Topology connection to %s successful\n", hostname) ;
 /* Try for Synchronous operation */
   if (nvotSetSynchronousCreation(ovwdbTimeout) == NVOT_SUCCESS)
   printf("wtotapi1: Open Topology calls will be synchronous") ;

  /* Main loop - read instructions and process until EOF */

 while(quit == FALSE)
  {
   strcpy(cmd, "") ;
   printf("Command> ") ;
   if( scanf("%s %[^\n]", cmd, args) == EOF) break ;
   switch (cmd_to_table(cmd))
    {
     case GTM_SET_PROTOCOL : set_cur_protocol(args) ; break ;
     case GTM_SET_PREFIX   : strcpy(oid_prefix, args) ; break ;
     case GTM_SET_PARENT   : set_cur_graph(args) ;    break ;
     case GTM_ADD_OBJECT   : add_sub_select(args) ;   break ;
     case GTM_SET_OBJECT   : set_vertex(args) ;       break ;
     default         : usage(GENERAL_USAGE) ;
    }
  }

 /* Finished with gtm connection */
 nvotDone() ;
}
```

Figure 185 (Part 9 of 9). wtotapi1.c Program Listing

C.2 wtgtm Shell Script Sample Listing

```
#!/usr/bin/ksh
#######################################
# usage
#######################################
function usage
{
   print "Usage for ${action[$1]}"
   print "          ${syntax[$1]}"

}

#######################################
# general_usage
#######################################
function general_usage
{
 clear
 print "************************************************************"
 print "NetView/6000 Generic topology Manager"
 print "************************************************************"
 count=1
 while ((count <= nbr_action))
 do
   print "$count: ${action[$count]}"
   print "   ${syntax[$count]}"
   ((count=count+1))
 done

 print "************************************************************"
}

#######################################
# Translate an icon
#######################################
function translate_icon
{
icon=$1
typeset -L1 first_char
first_char=$icon
case $first_char in
  1)      case $icon in # cards
            1:0) icon=".1.3.6.1.2.1.2.2.1.3.54.0";;
            1:1) icon=".1.3.6.1.2.1.2.2.1.3.11.1";;
            *)   icon=".1.3.6.1.2.1.2.2.1.3.54.0";;
          esac;;
  2)      case $icon in # computer
            2:1) icon=".1.3.6.1.4.1.2.2.1.2.3";;
            2:2) icon=".1.3.6.1.4.1.2.6.2";;
            2:3) icon=".1.3.6.1.2.1.2.2.1.3.9.10";;
            *)   icon=".1.3.6.1.4.1.2.2.1.2.3";;
          esac;;
  9)      case $icon in # network
            9:0) icon=".1.3.6.1.2.1.2.2.1.3.57.11";;
            9:1) icon=".1.3.6.1.2.1.2.2.1.3.9.11";;
            *)   icon=".1.3.6.1.2.1.2.2.1.3.57.11";;
          esac;;
  *)      print "Icon format is not valid, it should be x:y included in 1:0|1 2:1|2|3 9:0|1"
          exit;;
esac
}
```

Figure 186 (Part 1 of 6). wtgtm Shell Script Listing

```
#######################################
# Add a GRAPH in ROOT submap
#######################################
function add_root
{
protocol=$1
name=$2
layout=$4

translate_icon $3

snmptrap -c $COM_NAME $from_host .1.3.6.1.4.1.2 $to_host 6 1879048213 15 \
 .1.3.6.1.4.1.2.5.3.4.1.1.1 integer 3 \
 .1.3.6.1.4.1.2.5.3.4.1.1.2 objectidentifier .1.3.6.1.2.1.2.2.1.3.$protocol \
 .1.3.6.1.4.1.2.5.3.4.1.1.3 octetstring $name \
 .1.3.6.1.4.1.2.5.3.4.1.1.4 integer $layout \
 .1.3.6.1.4.1.2.5.3.4.1.1.10 objectidentifier $icon \
 .1.3.6.1.4.1.2.5.3.4.1.1.11 integer 1 \
 .1.3.6.1.4.1.2.5.3.4.1.1.12 octetstringascii $name
print "Trap <newGraph=$name> has been sent to gtmd"
}

#######################################
# Delete a GRAPH
#######################################
function delete_graph
{
protocol=$1
name=$2
snmptrap -c $COM_NAME  $from_host   .1.3.6.1.4.1.2.6.3.1  $to_host   6 1879048214 15 \
 .1.3.6.1.4.1.2.5.3.4.1.1.2 Objectidentifier .1.3.6.1.2.1.2.2.1.3.$protocol \
 .1.3.6.1.4.1.2.5.3.4.1.1.3 Octetstring $name
print "Trap <deleteGraph=$name> has been sent to gtmd"
}

#######################################
# Add a GRAPH in GRAPH Parent
#######################################
function add_graph
{
protocol=$1
name=$2
translate_icon $3
parent=$4
index=$5
snmptrap -c $COM_NAME $from_host .1.3.6.1.4.1.2 $to_host 6 1879048213 15 \
 .1.3.6.1.4.1.2.5.3.4.1.1.1 integer 4 \
 .1.3.6.1.4.1.2.5.3.4.1.1.2 objectidentifier .1.3.6.1.2.1.2.2.1.3.$protocol \
 .1.3.6.1.4.1.2.5.3.4.1.1.3 octetstring $name \
 .1.3.6.1.4.1.2.5.3.4.1.1.4 integer 7 \
 .1.3.6.1.4.1.2.5.3.4.1.1.10 objectidentifier $icon \
 .1.3.6.1.4.1.2.5.3.4.1.1.11 integer 2 \
 .1.3.6.1.4.1.2.5.3.4.1.1.12 octetstringascii $name
print "Trap <newGraph=$name> has been sent to gtmd"

snmptrap -c $COM_NAME $from_host .1.3.6.1.4.1.2 $to_host 6 1879048222 15 \
 .1.3.6.1.4.1.2.5.3.4.4.1.2 objectidentifier .1.3.6.1.2.1.2.2.1.3.$protocol \
 .1.3.6.1.4.1.2.5.3.4.4.1.3 octetstring $parent \
 .1.3.6.1.4.1.2.5.3.4.4.1.4 integer $index \
 .1.3.6.1.4.1.2.5.3.4.4.1.6 objectidentifier .1.3.6.1.2.1.2.2.1.3.$protocol \
 .1.3.6.1.4.1.2.5.3.4.4.1.7 octetstring $name
print "Trap <newMember $name in $parent> has been sent to gtmd"
}

#######################################
# Add a connection
#######################################
function connect_graph
{
protocol=$1
label=$2
from=$3
to=$4
parent=$5
arcid=$6
index=$7
```

Figure 186 (Part 2 of 6). wtgtm Shell Script Listing

```
snmptrap -c $COM_NAME $from_host .1.3.6.1.4.1.2 $to_host 6 1879048206 15 \
 .1.3.6.1.4.1.2.5.3.3.1.1.2 octetstringascii $label \
 .1.3.6.1.4.1.2.5.3.3.1.1.4 objectidentifier .1.3.6.1.2.1.2.2.1.3.$protocol \
 .1.3.6.1.4.1.2.5.3.3.1.1.5 octetstring $from \
 .1.3.6.1.4.1.2.5.3.3.1.1.7 objectidentifier .1.3.6.1.2.1.2.2.1.3.$protocol \
 .1.3.6.1.4.1.2.5.3.3.1.1.8 octetstring $to \
 .1.3.6.1.4.1.2.5.3.3.1.1.9 integer $arcid \
 .1.3.6.1.4.1.2.5.3.3.1.1.15 integer 2 \
 .1.3.6.1.4.1.2.5.3.3.1.1.16 integer 2 \
 .1.3.6.1.4.1.2.5.3.3.1.1.17 integer 0 \
 .1.3.6.1.4.1.2.5.3.3.1.1.18 integer 0
print "Trap <newArc $from $to> has been sent to gtmd"

snmptrap -c $COM_NAME $from_host .1.3.6.1.4.1.2 $to_host 6 1879048216 15 \
 .1.3.6.1.4.1.2.5.3.4.2.1.2 objectidentifier .1.3.6.1.2.1.2.2.1.3.$protocol \
 .1.3.6.1.4.1.2.5.3.4.2.1.3 octetstring $parent \
 .1.3.6.1.4.1.2.5.3.4.2.1.4 integer $index \
 .1.3.6.1.4.1.2.5.3.4.2.1.5 objectidentifier .1.3.6.1.2.1.2.2.1.3.$protocol \
 .1.3.6.1.4.1.2.5.3.4.2.1.6 octetstring $from \
 .1.3.6.1.4.1.2.5.3.4.2.1.7 objectidentifier .1.3.6.1.2.1.2.2.1.3.$protocol \
 .1.3.6.1.4.1.2.5.3.4.2.1.8 octetstring $to \
 .1.3.6.1.4.1.2.5.3.4.2.1.9 integer $arcid
print "Trap <memberArc $from $to in $parent> has been sent to gtmd"
}

#######################################
# Add an Underlying Arc parrallel
#######################################
function add_ula_parrallel
{
protocol=$1
fromg=$2
tog=$3
fromv=$4
tov=$5
arcid=$6
ulaid=$7
snmptrap -c $COM_NAME $from_host .1.3.6.1.4.1.2 $to_host 6 1879048210 15 \
 .1.3.6.1.4.1.2.5.3.3.2.1.2 objectidentifier .1.3.6.1.2.1.2.2.1.3.$protocol \
 .1.3.6.1.4.1.2.5.3.3.2.1.3 octetstring $fromg \
 .1.3.6.1.4.1.2.5.3.3.2.1.4 objectidentifier .1.3.6.1.2.1.2.2.1.3.$protocol \
 .1.3.6.1.4.1.2.5.3.3.2.1.5 octetstring $tog \
 .1.3.6.1.4.1.2.5.3.3.2.1.6 integer $arcid \
 .1.3.6.1.4.1.2.5.3.3.2.1.7 integer $ulaid \
 .1.3.6.1.4.1.2.5.3.3.2.1.8 integer 2 \
 .1.3.6.1.4.1.2.5.3.3.2.1.9 integer $protocol \
 .1.3.6.1.4.1.2.5.3.3.2.1.10 octetstring $fromv \
 .1.3.6.1.4.1.2.5.3.3.2.1.11 integer $protocol \
 .1.3.6.1.4.1.2.5.3.3.2.1.12 octetstring $tov \
 .1.3.6.1.4.1.2.5.3.3.2.1.13 integer $arcid
echo "trap sent to $0d"
print "Trap <underlyingArc $fromv $tov> has been sent to gtmd"
}


#######################################
# Add a vertex
#######################################
function add_vertex
{
protocol=$1
name=$2
translate_icon $3
parent=$4
index=$5

snmptrap -c $COM_NAME $from_host .1.3.6.1.4.1.2 $to_host 6 1879048192 15 \
 .1.3.6.1.4.1.2.5.3.1.1.1.2 integer $protocol \
 .1.3.6.1.4.1.2.5.3.1.1.1.3 octetstring $name \
 .1.3.6.1.4.1.2.5.3.1.1.1.9 objectidentifier $icon \
 .1.3.6.1.4.1.2.5.3.1.1.1.10 integer 2 \
 .1.3.6.1.4.1.2.5.3.1.1.1.11 integer 2 \
 .1.3.6.1.4.1.2.5.3.1.1.1.12 integer 0 \
 .1.3.6.1.4.1.2.5.3.1.1.1.13 integer 0
print "Trap <newVertex $name> has been sent to gtmd"
```

Figure 186 (Part 3 of 6). wtgtm Shell Script Listing

```
snmptrap -c $COM_NAME $from_host .1.3.6.1.4.1.2 $to_host 6 1879048222 15 \
 .1.3.6.1.4.1.2.5.3.4.4.1.2 objectidentifier .1.3.6.1.2.1.2.2.1.3.$protocol \
 .1.3.6.1.4.1.2.5.3.4.4.1.3 octetstring $parent \
 .1.3.6.1.4.1.2.5.3.4.4.1.4 integer $index \
 .1.3.6.1.4.1.2.5.3.4.4.1.6 integer $protocol \
 .1.3.6.1.4.1.2.5.3.4.4.1.7 octetstring $name
print "Trap <newMember $name in $parent> has been sent to gtmd"
}

#######################################
# Delete a vertex
#######################################
function del_vertex
{
protocol=$1
name=$2
snmptrap -c $COM_NAME  $from_host   .1.3.6.1.4.1.2.6.3.1  $to_host  6 1879048193 15 \
 .1.3.6.1.4.1.2.5.3.1.1.1.2 integer $protocol \
 .1.3.6.1.4.1.2.5.3.1.1.1.3 Octetstring $name
print "Trap <DeleteVertex $name> has been sent to gtmd"
}

#######################################
# add a Service Access Point (SAP)
#######################################
function add_sap
{
protocol1=$1
name1=$2
mode=$3
protocol2=$4
name2=$5
index=$6

case $mode in
  "1") modet="using";;
  "2") modet="providing";;
  "*") print"$mode must be 1 or 2"
       exit;;
esac

snmptrap -c $COM_NAME $from_host .1.3.6.1.4.1.2 $to_host 6 1879048196 15 \
 .1.3.6.1.4.1.2.5.3.1.2.1.2 integer $protocol1 \
 .1.3.6.1.4.1.2.5.3.1.2.1.3 octetstring $name1 \
 .1.3.6.1.4.1.2.5.3.1.2.1.4 integer $index \
 .1.3.6.1.4.1.2.5.3.1.2.1.5 integer $mode \
 .1.3.6.1.4.1.2.5.3.1.2.1.6 integer $protocol2 \
 .1.3.6.1.4.1.2.5.3.1.2.1.7 octetstring $name2

print "Trap <AddSAP $name1($protocol1) is $modet $name2($protocol2)> has been sent to gtmd"

}

#######################################
# delete a Service Access Point (SAP)
#######################################
function delete_sap
{
protocol=$1
name=$2

snmptrap -c $COM_NAME $from_host .1.3.6.1.4.1.2 $to_host 6 1879048197 15 \
 .1.3.6.1.4.1.2.5.3.1.2.1.2 integer $protocol \
 .1.3.6.1.4.1.2.5.3.1.2.1.3 octetstring $name \
 .1.3.6.1.4.1.2.5.3.1.2.1.4 integer 1
print "Trap <delSAP $name($protocol)> has been sent to gtmd"

}

#######################################
# set status for a vertex
#######################################
function set_status
{
protocol=$1
name=$2
status=$3
```

Figure 186 (Part 4 of 6). wtgtm Shell Script Listing

```
case $status in
  "down")
     snmptrap  -c $COM_NAME  $from_host .1.3.6.1.4.1.2.6.3.1  $to_host 6 1879048194 15 \
     .1.3.6.1.4.1.2.5.3.1.1.1.2 Integer $protocol \
     .1.3.6.1.4.1.2.5.3.1.1.1.3 Octetstring $name  \
     .1.3.6.1.4.1.2.5.3.1.1.1.9 Objectidentifier 1.3.6.1.2.1.2.2.1.3.54.0 \
     .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
     print "Trap <modifyVertex $name> has been sent to gtmd";;
  *) snmptrap  -c $COM_NAME  $from_host .1.3.6.1.4.1.2.6.3.1  $to_host 6 1879048194 15 \
     .1.3.6.1.4.1.2.5.3.1.1.1.2 Integer $protocol \
     .1.3.6.1.4.1.2.5.3.1.1.1.3 Octetstring $name  \
     .1.3.6.1.4.1.2.5.3.1.1.1.9 Objectidentifier 1.3.6.1.2.1.2.2.1.3.54.0 \
     .1.3.6.1.4.1.2.5.3.1.1.1.10 Integer 2 \
     .1.3.6.1.4.1.2.5.3.1.1.1.11 Integer 2 \
     .1.3.6.1.4.1.2.5.3.1.1.1.12 Integer 0 \
     .1.3.6.1.4.1.2.5.3.1.1.1.13 Integer 0
     print "Trap <modifyVertex $name> has been sent to gtmd";;
esac

}

#######################################
# Main Routine
#######################################
i=0
((i=i+1))
 action[$i]="Add a Graph in ROOT submap"
 syntax[$i]="$0 addr protocol name icon layout"
 ADDR=$i

((i=i+1))
 action[$i]="Add a Graph"
 syntax[$i]="$0 addg protocol name icon parent index"
 ADDG=$i

((i=i+1))
 action[$i]="Add a Vertex"
 syntax[$i]="$0 addv protoco] name icon parent index"
 ADDV=$i

((i=i+1))
 action[$i]="Add a Service Access point (SAP)"
 syntax[$i]="$0 sap protocol1 name1 using(1)/providing(2) protocol2 name2 index"
 SAP=$i

((i=i+1))
 action[$i]="Connection between two graphs"
 syntax[$i]="$0 cong protocol label from to parent arcid index"
 CONG=$i

((i=i+1))
 action[$i]="Add an underlying arc (parallel)"
 syntax[$i]="$0 ulap protocol graph1 graph2 vertex1 vertex2 arcid ulaid"
 ULAP=$i

((i=i+1))
 action[$i]="Delete a Graph"
 syntax[$i]="$0 delg protocol name"
 DELG=$i

((i=i+1))
 action[$i]="Delete a Vertex"
 syntax[$i]="$0 delv protocol name"
 DELV=$i

((i=i+1))
 action[$i]="Delete a Service Access point (SAP)"
 syntax[$i]="$0 dels protocol name"
 DELS=$i

((i=i+1))
 action[$i]="Set status for a vertex"
 syntax[$i]="$0 set protocol name up/down"
 SET=$i

nbr_action=i
```

Figure 186 (Part 5 of 6). wtgtm Shell Script Listing

```
from_host=`hostname`
to_host=`hostname`
COM_NAME="ITSC"

case $1 in

  "addr")
     case $2 in
         "?") usage $ADDR;;
         "") usage $ADDR;;
         *) add_root          $2 $3 $4 $5;;
     esac;;

  "addg")
     case $2 in
         "?") usage $ADDG;;
         "") usage $ADDG;;
         *) add_graph         $2 $3 $4 $5 $6;;
     esac;;

  "delg")
     case $2 in
         "?") usage $DELG;;
         "") usage $DELG;;
         *) delete_graph      $2 $3;;
     esac;;

  "addv")
     case $2 in
         "?") usage $ADDV;;
         "") usage $ADDV;;
         *) add_vertex        $2 $3 $4 $5 $6;;
     esac;;

  "delv")
     case $2 in
         "?") usage $DELV;;
         "") usage $DELV;;
         *) del_vertex        $2 $3;;
     esac;;

  "cong")
     case $2 in
         "?") usage $CONG;;
         "") usage $CONG;;
         *) connect_graph     $2 $3 $4 $5 $6 $7 $8;;
     esac;;

  "ulap")
     case $2 in
         "?") usage $ULAP;;
         "") usage $ULAP;;
         *) add_ula_parrallel $2 $3 $4 $5 $6 $7 $8;;
     esac;;

  "sap")
     case $2 in
         "?") usage $SAP;;
         "") usage $SAP;;
         *)  add_sap           $2 $3 $4 $5 $6 $7;;
     esac;;

  "dels")
     case $2 in
         "?") usage $DELS;;
         "") usage $DELS;;
         *) delete_sap        $2 $3;;
     esac;;

  "set")
     case $2 in
         "?") usage $SET;;
         "") usage $SET;;
         *) set_status          $2 $3 $4 $5;;
     esac;;

  *)      general_usage;;
esac
```

Figure 186 (Part 6 of 6). wtgtm Shell Script Listing

Appendix D. Database Samples

This appendix contains examples of code that exercises the SQL database support of NetView for AIX:

- wtqnode - shell script version

  A script that reports node and interface information based on the results of SQL queries of the IP topology database.

- wtqnode - C program example

  An enhanced version of the shell script, written in C with embedded SQL queries.

- wtqnetwork

  A C program that displays segment and interface information based on data from the IP topology database.

- wttraplog

  A C program that produces a report of Node Up/Node Down incidents with down-time.

- wtovwconv

  A program that converts NetView for AIX object database contents into SQL tables.

D.1 Sample Shell Script wtqnode

This sample was tested using the Oracle RDBMS.

```
#!/usr/bin/ksh

export ORACLE_SID=I
export ORACLE_HOME=/usr/oracle

WORK=wtqnode.out

sqlplus -s uid/pwd @lc $1 > $WORK

count=0
nbif=0
while read line
do
        (( count=count+1 ))
        if [ "$count" -le 3 ]
        then
           continue;
        else
           (( nbif=nbif+1 ))
           objid[$nbif]=`echo $line | cut -d" " -f1`
           tmp=`echo $line | cut -d" " -f2`
           iface[$nbif]=$tmp
        fi
done < $WORK

print "IP hostname:"
sqlplus -s uid/pwd @lo ${objid[1]} "ip_hostname" | awk '!/old | !/new &&
 length>0'

print "IP address:"
sqlplus -s uid/pwd @lo ${objid[1]} "snmpaddr" \
         | awk '!/old |new / && length>0'

print "=============================================================="
print "ObjID    IP Address      Phys.Address    Type IP Network"
print ""
count=1
while ((count < nbif))
do
   sqlplus -s uid/pwd @li1 ${iface[$count]} \
            | awk '!/old |new / && length>0'
   (( count=count+1 ))
done
```

Figure 187. wtqnode AIX Script File

The three SQL queries embedded by this script, lc.sql, lo.sql, and li1.sql, are shown below:

```
select nodeclass.objid,memberof.containedobjid
  from nodeclass,memberof
  where
    containerobjid =
    (select objid from nodeclass where ip_hostname like '%&1%')
  and
    nodeclass.objid=memberof.containerobjid
  ;
exit
```

Figure 188. lc.sql SQL Query File. This query extracts the object IDs of the node and all interfaces contained within it from the IP topology database.

```
select &2
  from nodeclass
  where objid=&1
  ;
exit
```

Figure 189. lo.sql SQL Query File. This very simple query takes two arguments. The first is an objectID that defines the node being accessed, the second is the name of the table column we want to extract for this row.

```
column objid format a8
column description format a15
column snmp_ifphysaddr format a15
column snmp_ifdescr format a4 truncated
column ip_network_name format a15
select interfaceclass.objid,
       interfaceclass.ip_address,
       snmp_ifphysaddr,
       snmp_ifdescr,
       ip_network_name
  from interfaceclass,
       coupledwith,
       objecttable,
       networkclass
 where interfaceclass.objid=&1
   and coupledwith.objid2=interfaceclass.objid
   and coupledwith.objid1=objecttable.objid
   and objecttable.classid=1
   and networkclass.objid=
       (select objid1
         from coupledwith,
              objecttable
        where coupledwith.objid2=&1
          and classid=1
          and coupledwith.objid1=objecttable.objid)
       ;
exit
```

Figure 190. li1.sql SQL Query File. This query extracts detailed interface information for a given interface object ID.

D.2 Sample C Program wtqnode

This sample was tested using the Informix RDBMS. The make file that we used to compile it follows the program listing.

```
/*-------------------------------------------------------------------*
 *                                                                   *
 *              wtqnode.ec Sample SQL Report Program                 *
 *                                                                   *
 *  (C) Copyright International Business Machines Corporation 1994   *
 *                                                                   *
 *   This program includes embedded SQL select statements, which     *
 *   extract details about a given node or nodes, including          *
 *   network interface data.  The program formats this data and      *
 *   prints it                                                       *
 *                                                                   *
 *   From redbook GG24-xxxx, Examples Using AIX NetView/6000 V3      *
 *                                                                   *
 *-------------------------------------------------------------------*/
#include <stdio.h>
$include sqltypes;

char errmsg[400];
char *convert_sqlstatus();
main(int argc, char *argv[])
{
   $char *s;
   $char node_id[36];
   $char ip_hostname[36];
   $long objid;
   $long interface_id;
   $char snmp_sysdescr[255];
   $char snmp_syslocation[255];
   $char snmp_syscontact[255];
   $char snmpaddr[15];
   $long topm_interface_count;
   $char ip_address[15];
   $char ip_network_name[15];
   $char snmp_ifdescr[5];
   $char snmp_ifphysaddr[14];
   $int  ip_status;
   char *c_ip_status;

   int count;
   if (strcmp(argv[1],"?") == 0 )
   {
      printf("Usage: %s [node_name].\n",argv[0]);
      printf("       This command list the nodes contained in Informix Database.\n");
      printf("       If [node_name] is specified then the list will be reduced to nodes\n");
      printf("       where ip_hostname matches with [node_name].\n");
      exit(1);
   }

   /* open openview database */
   $database openview;
   err_chk("Open database");

   /* build search value will be %node_name% */
   if(argc==1) sprintf(node_id,"%s","%");
          else sprintf(node_id,"%s%s%s","%",argv[1],"%");

   /* prepare and declare the two needed queries   */
   /* 1. q1,c1: list of nodes with objid           */
   /* 2. q2,c2: list of interfaces within a node   */

   /* prepare the first select for how many nodes
      and find out their objid */
   $prepare q1 from
         "select objid,
                 ip_hostname,
                 snmp_sysdescr,
                 snmp_syslocation,
                 snmp_syscontact,
                 snmpaddr
            from nodeclass
           where ip_hostname like ?;";
   err_chk("Prepare query q1");
```

Figure 191 (Part 1 of 4). wtqnode.ec C Program with Embedded SQL

```
/* declare a cursor in case of several rows */
$declare c1 cursor for q1;
err_chk("Declare Cursor c1");

/* list of interfaces */
$prepare q2 from
   "select containedobjid
      from nodeclass,
           memberof
     where containerobjid=?
      and  memberof.containerobjid=nodeclass.objid;";
err_chk("Prepare query q2");

/* declare a cursor for list of interfaces */
$declare c2 cursor for q2;
err_chk("Declare Cursor c2");

/* open cursor c1 */
$open c1 using $node_id;
err_chk("Open cursor c1");

/* */
count=0;
$fetch c1 into $objid,
               $ip_hostname,
               $snmp_sysdescr,
               $snmp_syslocation,
               $snmp_syscontact,
               $snmpaddr;
if(SQLCODE==SQLNOTFOUND)
{
   printf("No rows has been selected for criteria %s\n\n",node_id);
   exit(1);
}

while(1)
{
   if (count != 0)
   {
      $fetch c1 into $objid,
                     $ip_hostname,
                     $snmp_sysdescr,
                     $snmp_syslocation,
                     $snmp_syscontact,
                     $snmpaddr;
      if(SQLCODE==SQLNOTFOUND)  break;
   }
   count++;
   printf ("═══════════════════════════════════════════════════════════════════════════\n");
   printf ("objid:%d - ip_hostname: %s\n",objid,ip_hostname);
   printf ("Description : %s\n",snmp_sysdescr);
   printf ("Location    : %s\n",snmp_syslocation);
   printf ("Contact     : %s\n",snmp_syscontact);
   printf ("SNMP address: %s\n",snmpaddr);
   printf ("---------------------------------------------------------------------------\n");

   $open c2 using $objid;
   err_chk("Open cursor c2");
   printf("Objid    IP Address      Network Name    Phys. Address Type Status\n");
   printf("-------- --------------- --------------- ------------- ---- ---------\n");
```

Figure 191 (Part 2 of 4). wtqnode.ec C Program with Embedded SQL

```
      while(1)
      {
         $fetch c2 into $interface_id;
         if (SQLCODE == SQLNOTFOUND)  break; /* that means end of rows */

         /* give characteristics of interface including network address */
         $select interfaceclass.ip_address,
                 ip_network_name,
                 snmp_ifphysaddr,
                 snmp_ifdescr,
                 interfaceclass.ip_status
            into $ip_address,
                 $ip_network_name,
                 $snmp_ifphysaddr,
                 $snmp_ifdescr,
                 $ip_status
            from interfaceclass,
                 coupledwith,
                 objecttable,
                 networkclass
           where interfaceclass.objid=$interface_id
             and coupledwith.objid2=interfaceclass.objid
             and coupledwith.objid1=objecttable.objid
             and objecttable.classid=1
/*
*/
             and networkclass.objid =
                 (select objid1
/*
                    from interfaceclass,
                         coupledwith,
*/
                    from coupledwith,
                         objecttable
/*
                   where interfaceclass.objid=$interface_id
                     and coupledwith.objid2=interfaceclass.objid
*/
                   where coupledwith.objid2=$interface_id
                     and classid=1
                     and coupledwith.objid1=objecttable.objid);

          /* convert status in something comprehensible */
          c_ip_status = convert_sqlstatus(ip_status);

          printf("%7d ",interface_id);
          printf("%15s ",ip_address);
          printf("%15s ",ip_network_name);
          printf("%14s ",snmp_ifphysaddr);
          printf("%4s ",snmp_ifdescr);
          printf("%s ",c_ip_status);
          printf("\n");

      }
      $close c2;


   }
   printf ("═══════════════════════════════════════════════════════════════════════════════\n");
   printf("\n%d row(s) retrieved.\n",count);
}
```

Figure 191 (Part 3 of 4). wtqnode.ec C Program with Embedded SQL

```
/*-------------------------------------------------------------------------*/
/* convert the integer value of status                                     */
/*-------------------------------------------------------------------------*/
char * convert_sqlstatus(ip_status)
int ip_status;
{
   switch(ip_status)
   {
    case 1: return("unknown");
    case 2: return("up");
    case 3: return("marginal");
    case 4: return("down");
    case 5: return("unmanaged");
    case 6: return("acknowledge");
    case 7: return("User 1");
    case 8: return("User 2");
   }
}

/*-------------------------------------------------------------------------*/
/* err_chk() checks sqlca.code and if an error has occurred, it uses     */
/* rgetmsg() to display the message for the error number in sqlca.code.  */
/*-------------------------------------------------------------------------*/
err_chk(name)
char *name;
{
   if(sqlca.sqlcode != 0)
      {
      rgetmsg((short)sqlca.sqlcode, errmsg, sizeof(errmsg));
      printf("Error %d during %s: %s\n",sqlca.sqlcode, name, errmsg);
      exit(1);
      }
}
```

Figure 191 (Part 4 of 4). wtqnode.ec C Program with Embedded SQL

```
EXEC=wtqnode
SQL=$(EXEC).ec
SRC=$(EXEC).c
OBJ=$(EXEC).o
LIBS=-lgen -los -lsql -lbsd
FLAGS=-L/usr/informix/lib/esql
all: $(EXEC)

$(EXEC) : $(OBJ)
        cc -g -o $(EXEC) $(OBJ) $(LIBS) $(FLAGS)
$(OBJ) : $(SRC)
        cc -g -c $(SRC)
$(SRC) : $(SQL)
        esql -e $(SQL)
```

Figure 192. Makefile for wtqnode.ec Program Sample

D.3 Sample C Program wtqnetwork

This program was tested using the Informix RDBMS.

```
/*------------------------------------------------------------------------*
 *                                                                        *
 *           wtqnetwork.ec Sample SQL Report Program                      *
 *                                                                        *
 *  (C) Copyright International Business Machines Corporation 1994        *
 *                                                                        *
 *   This program includes embedded SQL select statements, which          *
 *   extract details about a given IP network or networks including       *
 *   all segments and interfaces. The program formats this data and       *
 *   prints it                                                            *
 *                                                                        *
 *   From redbook GG24-xxxx, Examples Using AIX NetView/6000 V3           *
 *                                                                        *
 *------------------------------------------------------------------------*/
#include <stdio.h>
$include sqltypes;

char errmsg[400];
char *convert_sqlstatus();
main(int argc, char *argv[])
{
   $char *s;
   $char network_name[36];
   $char snmp_ifphysaddr[15];
   $char ip_hostname[36];
   $char ip_network_name[36];
   $char n_ip_address[16];
   $char i_ip_address[16];
   $char selection_name[36];
   $char ip_network_address[16];
   $char ip_subnet_mask[16];
   $char i_ip_subnet_mask[16];
   $long i_objid;
   $long n_objid;
   $long segment_id;
   $int ip_status;
   char *c_ip_status;

   int count;
   int i_count;
   if (strcmp(argv[1],"?") == 0 )
   {
      printf("Usage: %s [network_name].\n",argv[0]);
      printf("       This command list the networks contained in Informix Database.\n");
      printf("       If [network_name] is specified then the list will be reduced to networks\n");
      printf("       where ip_network_name matches with [network_name].\n");
      exit(1);
   }

   /* open openview database */
   $database openview;
   err_chk("Open database");

   /* build search value will be %node_name% */
   if(argc==1) sprintf(network_name,"%s","%");
          else sprintf(network_name,"%s%s%s","%",argv[1],"%");

   /* prepare and declare the 3 needed queries     */
   /* 1. q1,c1: list of network                    */
   /* 1. q2,c3: list of segment in the network     */
   /* 1. q3,c3: list of interfaces in the segment  */

   /* list the network */
   $prepare q1 from
         "select objid,
                 ip_network_name,
                 ip_address,
                 ip_subnet_mask
            from networkclass
           where ip_network_name like ?;";
   err_chk("Prepare query q1");
```

Figure 193 (Part 1 of 3). wtqnetwork.ec Program Listing

```
/* declare a cursor for c1 */
$declare c1 cursor for q1;
err_chk("Declare Cursor c1");

/* list of segment in the network */
$prepare q2 from
   "select containedobjid,
           selection_name
      from memberof,
           segmentclass
     where containerobjid=?
       and memberof.containedobjid=segmentclass.objid;";
err_chk("Prepare query q2");

/* declare a cursor for c2 */
$declare c2 cursor for q2;
err_chk("Declare Cursor c2");

/* list of interfaces in the segment */
$prepare q3 from
    "select objid1,ip_address
       from coupledwith,
            interfaceclass
      where objid2=?
        and interfaceclass.objid=coupledwith.objid1
      order by ip_address;";
err_chk("Prepare query q3");

/* declare a cursor for c3 */
$declare c3 cursor for q3;
err_chk("Declare Cursor c3");


/* open cursor c1 */
$open c1 using $network_name;
err_chk("Open cursor c1");

/* */
count=0;
$fetch c1 into $n_objid,
               $ip_network_name,
               $n_ip_address,
               $ip_subnet_mask;
if(SQLCODE==SQLNOTFOUND)
{
   printf("No rows has been selected for criteria %s\n\n",network_name);
   exit(1);
}

while(1)
{
   if (count != 0)
   {
      $fetch c1 into $n_objid,
                     $ip_network_name,
                     $n_ip_address,
                     $ip_subnet_mask;
      if(SQLCODE==SQLNOTFOUND)  break;
   }
   count++;
   printf ("═══════════════════════════════════════════════════════════════════════════\n");
   printf ("OVw id = %d - Network name = %s\n",n_objid,ip_network_name);
   printf ("IP Address = %s - ",n_ip_address);
   printf ("Subnet mask = %s\n",ip_subnet_mask);

   $open c2 using $n_objid;
   err_chk("Open cursor c2");
   while(1)
   {
      $fetch c2 into $segment_id,
                     $selection_name;
      if (SQLCODE == SQLNOTFOUND)  break; /* that means end of rows */
      printf("--> Contents of Segment: %s\n",selection_name);
      printf ("---------------------------------------------------------------------\n");
      printf ("         ObjID  Stat IP Address           Phys. Address  Node\n");
      printf ("---------------------------------------------------------------------\n");
```

Figure 193 (Part 2 of 3). wtqnetwork.ec Program Listing

```
        $open c3 using $segment_id;
        err_chk("Open cursor c3");
        i_count = 0;
        while(1)
          {
               $fetch c3 into $i_objid;
               if (SQLCODE == SQLNOTFOUND)  break; /* that means end of rows */
               i_count++;
               /* find characteristics of interface including node name */
               $select ip_address,
                       snmp_ifphysaddr,
                       ip_hostname,
                       interfaceclass.ip_status
                  into $i_ip_address,
                       $snmp_ifphysaddr,
                       $ip_hostname,
                       $ip_status
                  from interfaceclass,
                       nodeclass,
                       memberof
                 where interfaceclass.objid=$i_objid
                   and memberof.containedobjid=interfaceclass.objid
                   and nodeclass.objid=memberof.containerobjid
                       ;
               c_ip_status=convert_sqlstatus(ip_status);
               printf("    %2d %5d %6s %14s ",i_count,i_objid,c_ip_status,i_ip_address);
               printf("    %14s %14s ",snmp_ifphysaddr,ip_hostname);
               printf("\n");

          }
        $close c3;

      }
      $close c2;
   }
   printf ("================================================================================\n");
   printf("\n%d row(s) retrieved.\n",count);
}

/*--------------------------------------------------------------------------*/
/* convert the integer value of status                                      */
/*--------------------------------------------------------------------------*/
char * convert_sqlstatus(ip_status)
int ip_status;
{
   switch(ip_status)
   {
    case 1: return("unknown");
    case 2: return("up");
    case 3: return("marginal");
    case 4: return("down");
    case 5: return("unmanaged");
    case 6: return("acknowledge");
    case 7: return("User 1");
    case 8: return("User 2");
   }
}

/*--------------------------------------------------------------------------*/
/* err_chk() checks sqlca.code and if an error has occured, it uses      */
/* rgetmsg() to display the message for the error number in sqlca.code.  */
/*--------------------------------------------------------------------------*/
err_chk(name)
char *name;
{
   if(sqlca.sqlcode != 0)
      {
      rgetmsg((short)sqlca.sqlcode, errmsg, sizeof(errmsg));
      printf("Error %d during %s: %s\n",sqlca.sqlcode, name, errmsg);
      exit(1);
      }
}
```

Figure 193 (Part 3 of 3). wtqnetwork.ec Program Listing

D.4 Sample C Program wttraplog

This program was tested using the Informix RDBMS.

```
/*-------------------------------------------------------------------*
 *                                                                   *
 *            wttraplog.ec Sample SQL Report Program                 *
 *                                                                   *
 *  (C) Copyright International Business Machines Corporation 1994   *
 *                                                                   *
 *   This program includes embedded SQL select statements, which     *
 *   extract Node Up and Node Down events from the trapdlog          *
 *   table and correlate them together. It prints a report showing   *
 *   node downtime.                                                  *
 *                                                                   *
 *   From redbook GG24-xxxx, Examples Using AIX NetView/6000 V3      *
 *                                                                   *
 *-------------------------------------------------------------------*/
#include <stdio.h>
$include sqltypes;

char errmsg·400“;
main(int argc, char *argv·“)
{
   $char ip_hostname·36“;
   $char description·255“;
   $char trap_create_time·20“;
   $long epochtime;

   /* Local variables to save Node Down information */
   char downname·36“ ;
   char downtime·20“ ;
   long down_epoch ;
   int count ;

   if (strcmp(argv·1“,”?”) == 0 )
   {
      printf(”Usage: %s \n”,argv·0“);
      printf(”       This command displays a report of node availability based  \n”);
      printf(”       on Node Down/Node Up records from the trapdlog table.\n”);
      printf(”       (Uses the informix database).\n”);
      exit(1);
   }

   /* open openview database */
   $database openview;
   err_chk(”Open database”);

   /* prepare and declare the query                          */
   /* q1,c1: list of Up/Down events, sorted by node and time*/

   $prepare q1 from
         ”select ip_hostname,
                 trap_create_time,
                 description,
                 epochtime
            from trapdlog
           where description like ’%Node Up%’
              or description like ’%Node Down%’
           order by ip_hostname,trap_create_time”;
   err_chk(”Prepare query q1”);

   /* declare a cursor in case of several rows */
   $declare c1 cursor for q1;
   err_chk(”Declare Cursor c1”);
```

Figure 194 (Part 1 of 2). wttraplog.ec Program Listing

```
   /* open cursor c1 */
   $open c1 ;
   err_chk("Open cursor c1");

   /* */
   count=0;
   $fetch c1 into $ip_hostname,
                  $trap_create_time,
                  $description,
                  $epochtime;
   if(SQLCODE==SQLNOTFOUND)
   {
      printf("No rows selected \n");
      exit(1);
   }
   printf ("=============================================================================================\n");
   printf("Node name                          Down from:         to:                Seconds:\n");
   printf("---------------------------------- ------------------ ------------------ --------\n");

   while(1)
   {
      if (count != 0)
      {
      $fetch c1 into $ip_hostname,
                     $trap_create_time,
                     $description,
                     $epochtime;
         if(SQLCODE==SQLNOTFOUND)  break;
      }
      count++;
         if (strncmp(description, "Node Down", 9) == 0 )
            {
             if (strcmp(downname, ip_hostname) != 0) printf("\n") ;
             strcpy(downtime,trap_create_time) ;
             strcpy(downname,ip_hostname) ;
             down_epoch = epochtime ;
            }
         else
            {
             if (strcmp(downname, ip_hostname) == 0)
                {
                 printf("%36s ", ip_hostname) ;
                 printf("%15s ", downtime) ;
                 printf("%15s ", trap_create_time) ;
                 printf("%d\n", epochtime-down_epoch) ;
                }
             }

   }
   printf ("=============================================================================================\n");
   printf("\n%d row(s) retrieved.\n",count);
}

/*---------------------------------------------------------------------------*/
/* err_chk() checks sqlca.code and if an error has occured, it uses      */
/* rgetmsg() to display the message for the error number in sqlca.code.  */
/*---------------------------------------------------------------------------*/
err_chk(name)
char *name;
{
   if(sqlca.sqlcode != 0)
      {
      rgetmsg((short)sqlca.sqlcode, errmsg, sizeof(errmsg));
      printf("Error %d during %s: %s\n",sqlca.sqlcode, name, errmsg);
      exit(1);
      }
}
```

Figure 194 (Part 2 of 2). wttraplog.ec Program Listing

D.5 Sample Program wtovwconv

This program generates SQL command files from the NetView for AIX object database.

```
/*-------------------------------------------------------------------*
 *                                                                   *
 *             wtovwconv.c Sample Program                            *
 *                                                                   *
 *  (C) Copyright International Business Machines Corporation 1994   *
 *                                                                   *
 *   This program uses OVw API calls to build sql statements that    *
 *   will create and load an SQL table containing the NV/6000        *
 *   object database contents.                                       *
 *                                                                   *
 *   From redbook GG24-xxxx, Examples Using AIX NetView/6000 V3      *
 *                                                                   *
 *-------------------------------------------------------------------*/
#include <stdio.h>
#include <OV/ovw_obj.h>
int warning;
int maxl=18;

usage()
{
   printf("Usage: wtovwconv table_name\n");
   printf("       table_name length has to be < 5\n");
   printf("       * will create wtc_'table_name'.sql SQL command file\n");
   printf("           - drop table wt_'table_name'\n");
   printf("           - create table wt_'table_name'\n");
   printf("       * will create wtl_'table_name'.sql SQL command file\n");
   printf("           - insert into table wt_'table_name'  for each OVwDb object\n");
   exit(1);
}

main(int argc,char *argv[])
{
FILE *f1;
FILE *f2;
OVwFieldBindList *fbl;
OVwObjectIdList *lo;
OVwFieldList *flp;
OVwFieldInfo *fip;
int i,j,ret;
int k,commit;
char out[256];
unsigned int flag=1;
char *tname[16];
char *f1name[16];
char *f2name[16];

if (argc!=2) usage();
if (strlen(argv[1]) > 4) usage();
sprintf(tname,"wt%s",argv[1]);
sprintf(f1name,"wtc_%s.sql",argv[1]);
sprintf(f2name,"wtl_%s.sql",argv[1]);

printf("* Connexion a OVwDb RC=%d\n",ret);
ret = OVwDbInit();
if(ret!=0)
  {
  exit(1);
  }

f1=fopen(f1name,"w");
f2=fopen(f2name,"w");

fprintf(f1,"drop table %s;\n\ncommit;\n\n",tname);
fprintf(f1,"create table %s (\nobjid integer primary key\n",tname);
/* list all fields */
flp = OVwDbListFields(ovwAllFields);

printf("* Building file %s command to create %s table.\n",f1name,tname);
```

Figure 195 (Part 1 of 4). wtovwconv.c Program Listing

```
for (i=0, fip=flp->field_list; i<flp->count; i++,fip++)
{
   if(fip->field_flags!=1)
   {
      grobw(fip->field_name,out);
      if (warning==0)
      {
      switch(fip->field_type)
      {
         /* fip->field_type = 1               */
         /* integer                          */
         case(1):
         {
             fprintf(f1,"%s%18s%s",
                       "'",
                       out,
                       " integer\n");
             break;
         }

         /* fip->field_type = 2               */
         /* boolean =char(5) false/true      */
         case(2):
         {
             fprintf(f1,"%s%18s%s",
                          "'",
                          out,
                          " char(5)\n");
             break;
         }

         /* fip->field_type = 3               */
         /* string                           */
         case(3):
         {
             fprintf(f1,"%s%18s%s",
                          "'",
                          out,
                          " char(255)\n");
             break;
         }

      /* fip->field_type = 4               */
      /* enum                             */
         case(4):
         {
             fprintf(f1,"%s%18s%s",
                          "'",
                          out,
                          " char(25)\n");
             break;
         }
       }
      }
   }
}

OVwDbFreeFieldList(flp);
fprintf(f1,");\n\ncommit;\n");
fclose(f1);

printf("* Building file %s command to load %s table.\n",f2name,tname);

/* do a commit each 20 insert */
commit = 20; /* commit is the limit */
k      = 1;  /* k is the counter */

/* get list of all objects */
lo=OVwDbListObjectsByFieldValue(NULL);
printf("  -> Loading %8d/%8d (Objid=%d)\n",1,lo->count,lo->object_ids[0]);
for (i=0; i < lo->count ; i++)
{
     fbl=OVwDbGetFieldValues(lo->object_ids[i]);
     /* new insert */
     fprintf(f2,"insert into %s (objid\n",tname);
```

Figure 195 (Part 2 of 4). wtovwconv.c Program Listing

```
/*----------------------------*/
/* 1. list the column to load */
/*----------------------------*/
for (j=0; j <fbl->count ; j++)
{
      /* get rid of list fields */
      if(fbl->fields[j].field_val->is_list==0)
      {
         grob(OVwDbFieldIdToFieldName(fbl->fields[j].field_id),out);
         if (warning==0) fprintf(f2,"           ,%s\n",out);
      }
}

/*----------------------------*/
/* 2. list the values to load */
/*----------------------------*/
fprintf(f2,"           )\nvalues(%d\n",
            lo->object_ids[i]);
for (j=0; j <fbl->count; j++)
{
   grob(OVwDbFieldIdToFieldName(fbl->fields[j].field_id),out);
   if (warning==0)
{
   /* get rid of list fields */
   if(fbl->fields[j].field_val->is_list==0)
   {
      switch(fbl->fields[j].field_val->field_type)
      {
         /* field_type=1 integer */
         case(1):
         {
            fprintf(f2,"        ,%d\n",
                        fbl->fields[j].field_val->un.int_val);
            break;
         }

         /* field_type=2 boolean false/true */
         case(2):
         {
            if(fbl->fields[j].field_val->un.bool_val==0)
               fprintf(f2,"       ,%cfalse%c\n",34,34);
            else
               fprintf(f2,"       ,%ctrue%c\n",34,34);
            break;
         }

         /* field_type=3 string */
         case(3):
         {
             grocr(fbl->fields[j].field_val->un.string_val,out);
             fprintf(f2,"       ,%c%s%c\n",34,
                         out,
                         34);
             break;
         }

         /* field_type=4 enum */
         case(4):
         {
             fprintf(f2,"       ,%d\n",
                         fbl->fields[j].field_val->un.enum_val);
             break;
         }

      } /* end of switch */
   }
}
}
fprintf(f2,"          );\n\n");

/* is it time to make a commit ? */
k++;
if (k==commit)
{
   k=1; /* initialize the counter */
   printf("  -> Loading %8d/%8d (Objid=%d)\n",i+1,lo->count,lo->object_ids[i]);
   fprintf(f2,"commit;\n\n");
}
```

Figure 195 (Part 3 of 4). wtovwconv.c Program Listing

```
     /* free memory for fields list */
     OVwDbFreeFieldList(fbl);
  }

  /* a last commit */
  printf("  -> Loading %8d/%8d (Objid=%d)\n",i,lo->count,lo->object_ids[i-1]);
  fprintf(f2,"commit;\n\n");

  /* close SQL insert file */
  fclose(f2);
}

grocr(char in[256],char out[256])
{
  int i=0;
  for(i=0; i<strlen(in); i++)
  {
     if(in[i] == '\n' )
     {
       out[i]=' ';
     }
     else out[i]=in[i];

  }
  out[i]='\0';
  return;
}

grob(char in[256],char out[256])
{
  int i=0,j=0;
  warning=0;
  out[0]=in[0];
  while(in[i] != '\0')
  {
    i++;
    if(in[i] != ' ' && in[i] != '.')
    {
       j++;
       out[j]=in[i];
       if(j==maxl+1)
       {
          j=j-1;
          warning=1;
          break;
       }
    }
  }
  out[j]='\0';
  return;
}

grobw(char in[256],char out[256])
{
  int i=0,j=0;
  warning=0;
  out[0]=in[0];
  while(in[i] != '\0')
  {
    i++;
    if(in[i] != ' ' && in[i] != '.')
    {
       j++;
       out[j]=in[i];
       if(j==maxl+1)
       {
          j=j-1;
          out[j]='\0';
          warning=1;
          printf("  -> Warning: column NOT loaded - length > %d - %s\n",maxl,in);
          return;
       }
    }
  }
  out[j]='\0';
  return;
}
```

Figure 195 (Part 4 of 4). wtovwconv.c Program Listing

Appendix E. NetView for AIX Default Events

```
# event name   number     catg       description
WARN_EV      0050462720    2   Warnings
NM_EV        0050790400    7   Node Marginal
SN_EV        0050790401    7   Segment Normal
SM_EV        0050790402    7   Segment Marginal
NETN_EV      0050790403    7   Network Normal
NETM_EV      0050790404    7   Network Marginal
SA_EV        0050790405    7   Segment Added
SD_EV        0050790406    7   Segment Deleted
NETA_EV      0050790407    7   Network Added
NETD_EV      0050790408    7   Network Deleted
CPP_EV       0050790411    7   Change Polling Period
FP_EV        0050790412    7   Forced Poll
MNET_EV      0050790416    7   Manage Network
UNET_EV      0050790417    7   Unmanage Network
MN_EV        0050790418    7   Manage Node
UN_EV        0050790419    7   Unmanage Node
MSEG_EV      0050790420    7   Manage Segment
USEG_EV      0050790421    7   Unmanage Segment
NMTM_EV      0050790423    7   Netmon Change trace mask
CIS_EV       0050790427    7   Change Interface Segment
FMTCHG       0050790438    7   trapd.conf format changed
MIBCHG       0050790439    7   ASN.1 mib definition file format changed
COLCHG       0050790440    7   SNMP data collector file format changed
MI_EV        0050790441    7   Manage Interface
UI_EV        0050790442    7   Unmanage Interface
NETFC_EV     0050790443    7   Network Flags changed
SEGFC_EV     0050790444    7   Segment Flags changed
NFC_EV       0050790445    7   Node Flags changed
IFC_EV       0050790446    7   Interface Flags changed
CPUL_EV      0058720256    0   CPU Load
DSPU_EV      0058720257    0   Disk Space Percentage Used
IPD_EV       0058720258    0   Interface Percent Deferred
IPC_EV       0058720259    0   Interface Percent Collisions
ICE_EV       0058720260    0   Interface CRC Errors
IPIE_EV      0058720261    0   Interface Percent Input Errors
IPOE_EV      0058720262    0   Interface Percent Output Errors
DCOL_EV      0058720263    0   Data Collector detected threshold
DCRA_EV      0058720264    0   Data Collector re-arm event
IADD_EV      0058785792    1   Interface Added
IDEL_EV      0058785793    1   Interface Deleted
NADD_EV      0058785794    1   Node Added
NDEL_EV      0058785795    1   Node Deleted
ERR_EV       0058851329    2   Non Fatal Errors
FERR_EV      0058851330    2   Fatal Errors
```

Figure 196 (Part 1 of 2). NetView for AIX event - I (Sorted by Event Number) August 1994

```
NUP_EV       0058916864    3   Node Up
NDWN_EV      0058916865    3   Node Down
IUP_EV       0058916866    3   Interface Up
IDWN_EV      0058916867    3   Interface Down
SC_EV        0058916868    3   Segment Critical
NC_EV        0058916869    3   Network Critical
SNMP_EV      0058916871    3   SNMP Status Event
LLAC_EV      0058982400    4   Link Level Address Changed
MLLA_EV      0058982401    4   Mismatch of Link Level Address
ULLA_EV      0058982402    4   Undetermined Link Level Address
OIC_EV       0058982403    4   Object Identifier Change
SDC_EV       0058982404    4   System Descr Change
SNC_EV       0058982405    4   System Name Change
SMC_EV       0058982406    4   Subnet Mask Change
FSC_EV       0058982407    4   Forwarding status change
FTH_EV       0058982408    4   Forwarding to a host
SCC_EV       0058982410    4   System Contact Change
SLC_EV       0058982411    4   System Location Change
ITC_EV       0058982412    4   Interface Type Change
IDC_EV       0058982413    4   Interface Descr Change
BSM_EV       0058982414    4   Bad Subnet Mask
AA_EV        0059047936    5   Application Alert
APUP_EV      0059179056    7   Application Up Event
APDN_EV      0059179057    7   Application Down Event
TATM_EV      0059179068    7   Tralert change tracemask Event
NMCR_EV      0059179070    7   Change netmon retry count
```

Figure 196 (Part 2 of 2). NetView for AIX event - I (Sorted by Event Number) August 1994

Appendix F. Nvevents X11 app-defaults File

```
!
! defines maximum number of events displayed by the application
!
nvevents.maxNumEvents         : 500
!
! defines maximum number of concurrent opened workspaces
!
nvevents.maxNumWS             : 20
!
! defines maximum number of events to be loaded from ovevent.log
!
nvevents.maxLoadEvents        : 500
!
! defines initial presentation style (card or list)
!
nvevents.initialPresCard      : True
!
! defines if workspace name is located at right or left of the window
!
nvevents.posRightName         : True
!
! directory used when saving the last active filters
!
nvevents.profileDir           : $HOME
!
! directory used when reading filters for filling the filter control window
!
nvevents.filterDir            : /usr/OV/filters/filter.samples
!
! directory used when creating reports
!
nvevents.reportDir            : $HOME
!
! directory used when saving workspaces
!
nvevents.saveDir              : $HOME
!
! size of nvevents windows
!
nvevents.widthMain            : 800
!
! size of the nvevents windows
!
nvevents.heightMain           : 500
!
! normal card color
!
nvevents.cardColor            : lightblue
!
! color of the card when it is selected
!
nvevents.cardColorSelect      : #ffdab9
```

Figure 197 (Part 1 of 2). /usr/lpp/X11/lib/X11/app-defaults/Nvevents Sample

```
!
! initial position of scroll bar in the card deck
!
nvevents.scrollBarUp          : False
!
! defines number of cards to "card deck" appearance
!
nvevents.numberFillCards      : 4
!
! controls double click interval in selecting items in the list
!
nvevents.doubleClickInterval  : 350
!
! color used as background in nvevents application
!
nvevents*background           : gray
!
! type of icon used in the nvevents (0: gif)
!
nvevents.iconType             : 0
!
! color to be used when writing icon label
!
nvevents.iconLabelColor       : black
!
! label for nvevents windows
!
nvevents.iconLabel            : Events
!
! icon used in the nvevents application shells
!
nvevents.iconBackground       : /usr/OV/icons/gifs/dynamic_events.gif
!
! font list used in the nvevents application
!
nvevents*FontList             : Rom10
!
! font to be used by the Card widget to write normal texts
!
nvevents*card*cardFontList    : Rom10
!
! font to be selected by the Card widget when writing small texts
!
nvevents*card*smallFontList   : tnrR10
!
! defines if application starts up outside of the Control Desk
! valid when running integrated to OVw
!
nvevents.outside              : False
!
! defines color to be used in the text written in the cards
!
nvevents*card*cardTextColor   : black
!
! defines foreground color to be used in the events application
!
nvevents*foreground           : black
!
! defines if log only events forces the execution of associated commands
!
nvevents.executeLogOnly       : True
!
! defines if new workspaces are opened outside the Control Desk
!
nvevents.wsOutside            : False
```

Figure 197 (Part 2 of 2). /usr/lpp/X11/lib/X11/app-defaults/Nvevents Sample

Appendix G. Selected AIX SNA Server Profiles

The current version of AIX NetView Service Point does not require extensive LU 6.2 definitions at the RISC System/6000. The following are some key parameters that this project used.

```
Change/Show Token-Ring SNA DLC Profile

[TOP]                                                  [Entry Fields]
  Current profile name                                 RS03ATT2
  New profile name                                    []
  Data link device name                               [tok0]
  Force disconnect time-out (1-600 seconds)           [120]
  User-defined maximum I-Field size?                   no
     If yes, Max. I-Field size (265-30729)            [30729]
  Max. num of active link stations (1-255)            [32]
     Number reserved for inbound activation           [0]
     Number reserved for outbound activation          [0]
  Transmit window count (1-127)                       [16]
  Dynamic window increment (1-127)                    [1]
  Retransmit count (1-30)                             [8]
  Receive window count (1-127)                        [8]
  Ring access priority                                 0
  Inactivity time-out (1-120 seconds)                 [120]
  Response time-out (1-40, 500 msec intervals)        [4]
  Acknowledge time-out (1-40, 500 msec intervals)     [1]
  Local link name                                     [RS03TOK0]
  Local SAP address (02-fa)                           [04]
  Trace base listening link station?                   yes
     If yes, Trace format                              long
  Dynamic link stations supported?                     yes

  Link Recovery Parameters
     Retry interval (1-10000 seconds)                 [60]
     Retry limit (0-500 attempts)                     [20]

  Dynamic Link Activation Parameters
     Solicit SSCP sessions?                            yes
     CP-CP sessions supported?                         yes
     Partner required to support CP-CP sessions?       no

  Dynamic Link TG COS Characteristics
     Effective capacity                               [4300800]
     Cost per connect time                            [0]
     Cost per byte                                    [0]
     Security                                          nonsecure
     Propagation delay                                 lan
     User-defined 1                                   [128]
     User-defined 2                                   [128]
     User-defined 3                                   [128]
  Comments                                            []
F1=Help        F2=Refresh         F3=Cancel            F4=List
F5=Reset       F6=Command         F7=Edit              F8=Image
```

Figure 198. Token Ring SNA DLC Profile

```
                                        Change/Show SNA Node Profile

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

                                                          [Entry Fields]
  Profile name                                            sna
  Maximum number of sessions (1-5000)                    [200]
  Maximum number of conversations (1-5000)               [200]
  Restart action                                          once
  Recovery resource manager (RRM) enabled?                no
  Dynamic inbound partner LU definitions allowed?         yes
  NMVT action when no NMVT process                        reject
  Standard output file/device                            [/dev/console]
  Standard error file/device                             [/dev/console]

  Comments                                               []
```

Figure 199. SNA Node Profile

```
                  Change/Show Token Ring Link Station Profile

Type or select values in entry fields.
Press Enter AFTER making all desired changes.
                                                        [Entry Fields]
  Current profile name                                  RA60003
  New profile name                                     []
  Use Control Point's XID node ID?                      yes                    +
     If no, XID node ID                                [*]
* SNA DLC Profile name                                 [RS03ATT2]             +
  Stop link station on inactivity?                      no                     +
     If yes, Inactivity time-out (0-10 minutes)        [0]                     #
  LU address registration?                              yes                    +
     If yes, LU Address Registration Profile name      [RA60003]              +
  Trace link?                                           yes                    +
     If yes, Trace size                                 long                   +

  Adjacent Node Address Parameters
     Access routing                                     link_address           +
     If link_name, Remote link name                    []
     If link_address,
        Remote link address                            [400002070000]          X
        Remote SAP address (02-fa)                     [04]                    X

  Adjacent Node Identification Parameters
     Verify adjacent node?                              no                     +
     Network ID of adjacent node                       [USIBMRA]
     CP name of adjacent node                          [RAPAN]
     XID node ID of adjacent node (LEN node only)      [*]

  Link Activation Parameters
     Solicit SSCP sessions?                             yes                    +
     Initiate call when link station is activated?      yes                    +
     Activate link station at SNA start up?             no                     +
     Activate on demand?                                no                     +
     CP-CP sessions supported?                          yes                    +
     If yes,
        Adjacent network node preferred server?         no                     +
     Partner required to support CP-CP sessions?        no                     +
     Initial TG number (0-20)                          [0]                     #

  Restart Parameters
     Restart on activation?                             no                     +
     Restart on normal deactivation?                    no                     +
     Restart on abnormal deactivation?                  no                     +

  Transmission Group COS Characteristics
     Effective capacity                                [4300800]               #
     Cost per connect time                             [0]                     #
     Cost per byte                                     [0]                     #
     Security                                           nonsecure              +
     Propagation delay                                  lan                    +
     User-defined 1                                    [128]                   #
     User-defined 2                                    [128]                   #
     User-defined 3                                    [128]                   #
  Comments                                             []
F1=Help            F2=Refresh         F3=Cancel          F4=List
F5=Reset           F6=Command         F7=Edit            F8=Image
F9=Shell           F10=Exit           Enter=Do
```

Figure 200. Link Station Profile

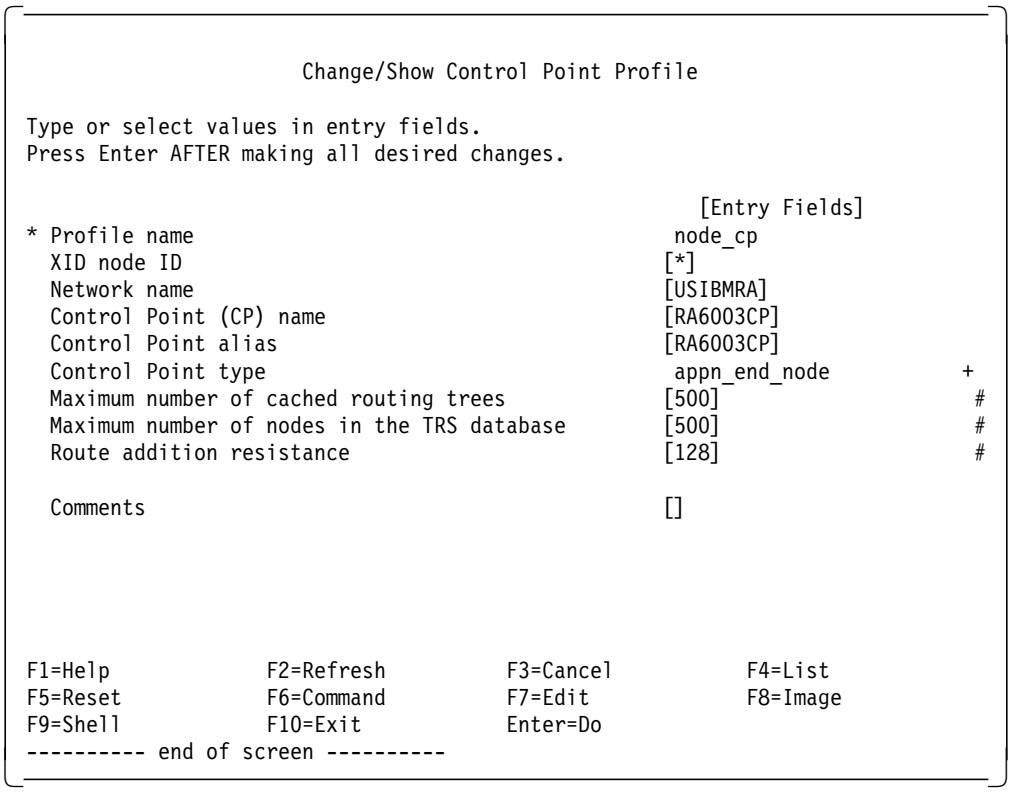


Figure 201. Control Point Profile

G.1 Selected S/390 VTAM Members

The connection for this project was token-ring.

```
*********************************************************************
*                                                                   *
*         VTAM SWITCHED MAJOR NODE FOR NTRI                         *
*   62222 RS/60003 OFFICE NUMBER BB110                              *
*********************************************************************
RA2RSKY VBUILD MAXGRP=10,             REQUIRED                    * X
               MAXNO=18,              REQUIRED                    * X
               TYPE=SWNET             REQUIRED
**
**
*** RA60003 IS FOR RS60003
*             : CPNAME IS USED ON RS60003 INSTEAD OF IDBLK AND IDNUM
*
*  IDBLK AND IDNUM USED TO BE:
*              IDBLK=05D,             PC 3274 EMULATOR            *
*              IDNUM=62222,           SA 20, 3174 TYPE, FIRST 3174  *
*
RA60003  PU    ADDR=13,               COULD BE ANYTHING (NOT USED)  * X
               CPNAME=RA6003CP,       USED FOR AIX NETVIEW SP       * X
               MODETAB=AMODETAB,                                    * X
               MAXPATH=2,                                           * X
               MAXDATA=265,                                           *
               MAXOUT=7,                                              *
               PACING=7,                                              *
               ANS=CONTINUE,                                          *
               PASSLIM=7,                                             *
               PUTYPE=2,                                              *
               DISCNT=(NO),                                           *
               ISTATUS=ACTIVE,                                        *
               VPACING=8
**
RA600030 LU    LOCADDR=0,MODETAB=MODEVR,DLOGMOD=M2SDLCQ
RA60003B LU    LOCADDR=0,MODETAB=AMODEAPP,DLOGMOD=NVDMNORM
RA60003V LU    LOCADDR=0,MODETAB=MODEVR,DLOGMOD=M2SDLCQ
RA60003W LU    LOCADDR=0,MODETAB=MODEVR,DLOGMOD=M2SDLCQ
RA60003X LU    LOCADDR=0,MODETAB=MODEVR,DLOGMOD=M2SDLCQ
RA60003Y LU    LOCADDR=0,MODETAB=MODEVR,DLOGMOD=M2SDLCQ
RA60003Z LU    LOCADDR=0,MODETAB=AMODETAB,DLOGMOD=DSIL6MOD
```

Figure 202 (Part 1 of 2). Switched Major Node Definition Used In This Example

```
RA600032 LU    LOCADDR=2,                                              *
               MODETAB=AMODESHO,                                       *
               DLOGMOD=AIXLGMD2,                                       *
               ISTATUS=ACTIVE
RA600033 LU    LOCADDR=3,                                              *
               MODETAB=MODNDM12,                                       *
               DLOGMOD=AIXLGMD1,                                       *
               ISTATUS=ACTIVE
RA600034 LU    LOCADDR=4,                                              *
               MODETAB=MODNDM12,                                       *
               DLOGMOD=AIXLGMD1,                                       *
               ISTATUS=ACTIVE
RA600035 LU    LOCADDR=5,                                              *
               MODETAB=MODNDM12,                                       *
               DLOGMOD=AIXLGMD1,                                       *
               ISTATUS=ACTIVE
RA600036 LU    LOCADDR=6,USSTAB=US327X,SSCPFM=USSSCS,MODETAB=AMODETAB, X
               DLOGMOD=M2SDLCQ
RA600037 LU    LOCADDR=7,USSTAB=US327X,SSCPFM=USSSCS,MODETAB=AMODETAB, X
               DLOGMOD=M2SDLCQ
* CHANGED RA600038/9 MODETABLES,ENTRIES FOR HCON - BDN 11/10/92
RA600038 LU    LOCADDR=8,USSTAB=US327X,SSCPFM=USSSCS,MODETAB=AMODHCON, X
               DLOGMOD=LU1HCON
RA600039 LU    LOCADDR=9,USSTAB=US327X,SSCPFM=USSSCS,MODETAB=AMODHCON, X
               DLOGMOD=LU3HCON
RA60003C LU    LOCADDR=12,MODETAB=MODEVR,DLOGMOD=M3SDLCQ
```

Figure 202 (Part 2 of 2). Switched Major Node Definition Used In This Example. The LUs defined above were used by applications other than the Service Point. The AIX SNA Server is keying on the CPNAME for Service Point functions and an LU need not be defined.

```
         VBUILD TYPE=CDRSC
         NETWORK NETID=USIBMRA
*                          SERVICE POINT FOR RS60003
RA6003CP CDRSC   ALSLIST=(RA60003)
*
```

Figure 203. CDRSC Definition Used In This Example

The above resource, for previous versions of AIX NetView Service Point, was a resource in the switched major node PU definition as:

```
RA6003CP  LU  LOCADDR=0
```

When using CPNAME and current level of VTAM is being exploited, it is not necessary to include the LOCADDR=0 in the switched major node. This project chose to use the above CDRSC and activated the CDRSC prior to activation of the RISC System/6000 link station. The ″old″ LOCADDR=0 specification was removed from the switched major node definition.

Index

Special Characters

A

B

C

D

E

G

H

I

L

M

N

U

V

W

X

ITSO Technical Bulletin Evaluation RED000

International Technical Support Organization
Examples of
Using NetView for AIX
November 1994

Publication No. GG24-4327-00

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction	____		
Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) If you are an employee of IBM or its subsidiaries:

Do you provide billable services for 20% or more of your time? Yes____ No____

Are you in a Services Organization? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

_____

_____

What other topics would you like to see in this Bulletin?

_____

_____

What other Technical Bulletins would you like to see published?

_____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

Name	Address
Company or Organization	
Phone No.	

Cut or Fold Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Department 545, Building 657
P.O. BOX 12195
RESEARCH TRIANGLE PARK NC
USA 27709-2195

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold Along Line

Printed in U.S.A.