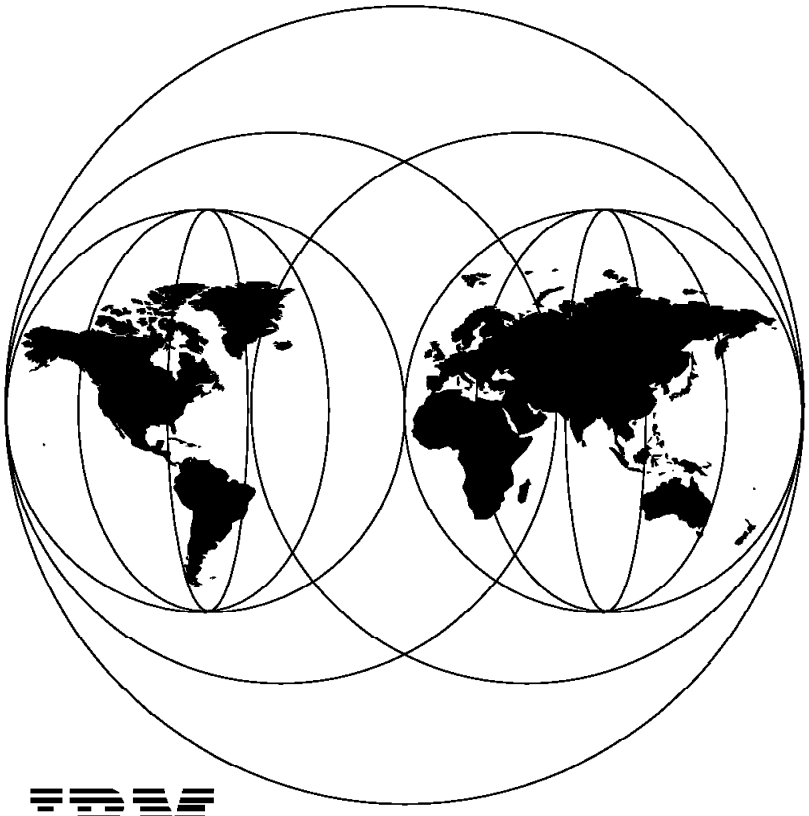
SG24-4504-01

Introduction to IBM AS/400 SNMP Support

November 1997



IBM

International Technical Support Organization
Raleigh Center

International Technical Support Organization

SG24-4504-01

Introduction to IBM AS/400 SNMP Support

November 1997

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix G, “Special Notices” on page 299.

Second Edition (November 1997)

This edition applies to the licensed program IBM Operating System/400, Version 3 Release 2 (Program 5763-SS1) and OS/400 Version 3 Release 6 (Program 5716-SS1) but the book contents are applicable through OS/400 V4R1.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 678
P.O. Box 12195
Research Triangle Park, NC 27709-2195

Contents

Preface ix
The Team That Wrote This Redbook ix
Comments Welcome x

Chapter 1. Introduction to SNMP 1
1.1 Managing a Heterogeneous Network 1
1.2 A Brief View into SNMP History 2
1.3 Protocol Layering 3
1.3.1 The OSI Reference Model 3
1.3.2 Time Sequence Diagrams 4
1.3.3 The TCP/IP Protocol Suite Model 5
1.4 SNMP Definitions 6

Chapter 2. Simple Network Management Protocol (SNMP) 9
2.1 The SNMP Architecture 9
2.2 Goals of the SNMP Architecture 9
2.3 SNMP Model 10
2.4 User Datagram Protocol (UDP) 10
2.5 Asynchronous Request/Response Protocol 11
2.6 SNMP Agent 12
2.7 SNMP Subagent 13
2.8 SNMP Manager 13
2.9 Understanding MIBs 14
2.9.1 Representation of Management Information 15
2.9.2 MIB Naming Conventions 16
2.10 SNMP Operations 20
2.11 Table Traversal 21

Chapter 3. OS/400 SNMP Support 23
3.1 OS/400 SNMP Agent 24
3.1.1 Standard RFC MIBs 24
3.1.2 IBM Enterprise MIBs 26
3.1.3 Novell Enterprise MIBs 26
3.1.4 DPI 2.0 27
3.1.5 SNMP Subagent MIB 27
3.1.6 Trap Support 27
3.2 OS/400 SNMP Manager 27
3.3 OS/400 SNMP Subagent 28
3.4 OS/400 SNMP General Overview 29

Chapter 4. Getting Started 31
4.1 TCP/IP Considerations 31
4.2 Verifying a TCP/IP Connection 31
4.3 Configuring the AS/400 SNMP Agent 34
4.3.1 Configuring SNMP Attributes 34
4.3.2 Configuring SNMP Communities 37
4.3.3 Starting the AS/400 SNMP Agent 40

Chapter 5. NetView for OS/2 41
5.1 Using NetView for OS/2 as an SNMP Manager 41
5.1.1 The NetView for OS/2 Desktop 41

5.1.2 The Discovery Process 44
5.1.3 Using NetView for OS/2 to Retrieve Data 46
5.1.4 Using the MIB Browser 50
5.1.5 The NetView for OS/2 Grapher Function 57
5.1.6 Using NetView for OS/2 to Set Information 61
5.1.7 Managing Multiple AS/400s 65

Chapter 6. OS/400 SNMP Manager 75
6.1 SNMP Manager APIs Overview 75
6.1.1 SNMP Manager API Functions 76
6.1.2 SNMP Manager Example 77
6.2 Trap Manager Overview 78
6.2.1 Trap Forwarding 80
6.2.2 Traps Delivered to Data Queues 84

Chapter 7. OS/400 SNMP Subagent 91
7.1 Introduction to SNMP Distributed Protocol Interface 91
7.2 SNMP Agent and Subagents 91
7.3 DPI Agent Requests 92
7.4 Description of SNMP Subagent DPI Requests 97
7.4.1 Subagent Programming Concepts 97
7.4.2 SNMP Agent Initiated DPI Requests 97
7.4.3 SNMP Subagent Initiated DPI Requests 103
7.4.4 SNMP Subagent Communication Functions with SNMP Agent . . . 104
7.5 SNMP Subagent APIs 105
7.5.1 Subagent API Functions 106
7.6 SNMP Subagent Example 107

Chapter 8. OS/400 SNMP-Based Functions 127
8.1 Client Inventory Management 127
8.1.1 Client Software Management Database Formats 129
8.1.2 The Client Management MIB 137
8.1.3 Browsing the Client Management MIB 139
8.1.4 Client Access/400 Optimized for OS/2 145

Chapter 9. OS/400 Supported MIBs 147
9.1 Standard RFC MIBs 148
9.1.1 MIB-II 148
9.1.2 Host (Host Resources) MIB 150
9.1.3 Ethernet-Like Interface MIB 150
9.1.4 Browsing the Transmission Group MIB 151
9.1.5 FDDI MIB 153
9.1.6 Frame Relay MIB 154
9.1.7 IEEE 802.5 Token-Ring MIB 155
9.2 DPI 2.0 MIB 158
9.3 SNMP Subagent MIB 158
9.4 IBM Enterprise MIBs 159
9.4.1 Advanced Peer-To-Peer Networking (APPN) MIB 159
9.4.2 Client Management MIB 170
9.4.3 NetView for AIX Subagent MIB 171
9.4.4 AS/400 Remote Workstation MIB 171
9.5 Novell Enterprise MIBs 178
9.5.1 Internetwork Packet Exchange (IPX) MIB 178
9.5.2 Routing Information and Service Advertising Protocols MIB 186
9.5.3 NetWare Link Services Protocol (NLSP) MIB 187

Chapter 10. Host Resources MIB . . . 189
10.1 Host Resources MIB and AS/400 . . . 189
10.1.1 Browsing the Host Resources MIB on AS/400 . . . 190
10.2 Host Resources MIB and Client Access Optimized for OS/2 . . . 197
10.2.1 Browsing the Client Access Host Resources MIB . . . 199

Chapter 11. AS/400 as a Trap Generator . . . 205
11.1 Configuration Requirements . . . 205
11.2 Examples of the AS/400 Generated Traps . . . 206
11.2.1 The ColdStart Trap . . . 206
11.2.2 LinkDown and LinkUp Traps . . . 207
11.2.3 The AuthenticationFailure Trap . . . 208

Chapter 12. Journal for SNMP Logging . . . 211
12.1 QSNMP Journal Overview . . . 211
12.2 Examples of Journals Entries . . . 213

Chapter 13. Loading an Enterprise-Specific MIB . . . 219
13.1 OS/400 and IBM Enterprise MIBs . . . 219
13.2 Transferring the MIB Module to the Manager . . . 220
13.2.1 Transferring a MIB Module Using FTP . . . 221
13.2.2 Loading the MIB into NetView for OS/2 . . . 224

Chapter 14. SNMP and AnyNet . . . 227
14.1 AnyNet Overview . . . 227
14.1.1 APPC over TCP/IP . . . 227
14.1.2 Sockets over SNA . . . 228
14.1.3 Multiprotocol Transport Networking (MPTN) Architecture . . . 228
14.1.4 AnyNet/400 Considerations . . . 229
14.2 SNMP AnyNet Scenarios . . . 230

Chapter 15. NetView for AIX . . . 231
15.1 Overview . . . 231
15.2 NetView for AIX Graphical Interface . . . 231

Appendix A. Additional Information on SNMP . . . 235
A.1 The MIB Tree Structure . . . 235
A.1.1 Directory Subtree . . . 236
A.1.2 Mgmt Subtree . . . 236
A.1.3 Experimental Subtree . . . 237
A.1.4 Private Subtree . . . 237
A.2 Administrative SNMP Terminology . . . 238
A.3 SNMP Messages . . . 240
A.3.1 How SNMP Messages Are Processed . . . 240
A.3.2 SNMP Protocol Data Units . . . 241

Appendix B. The IAB . . . 245
B.1 The Internet Activities Board (IAB) . . . 245
B.1.1 Request for Comments (RFC) . . . 246
B.1.2 Functions of the IAB . . . 247
B.1.3 Protocol Standardization Process . . . 247

Appendix C. Problem Determination . . . 251
C.1 Problem Determination for Starting OS/400 SNMP . . . 251
C.2 Problem Determination for SNMP Agent . . . 254

C.3 Problem Determination for SNMP Manager APIs 259
C.4 Problem Determination for SNMP Trap Manager 262
C.5 Problem Determination for SNMP Subagent APIs 264

Appendix D. CL Commands for AS/400 SNMP 267
D.1 ADDCOMSNMP (Add Community for SNMP) Command 267
D.1.1 Purpose 267
D.1.2 Required Parameter 268
D.1.3 Optional Parameters 268
D.2 CFGTCPSNMP (Configure TCP/IP SNMP) Command 271
D.2.1 Purpose 271
D.3 CHGCOMSNMP (Change Community for SNMP) Command 273
D.3.1 Purpose 273
D.3.2 Required Parameter 273
D.3.3 Optional Parameters 273
D.4 CHGSNMPA (Change SNMP Attributes) Command 276
D.4.1 Purpose 276
D.4.2 Optional Parameters 277
D.5 ENDTCPSVR (End TCP/IP Server) Command 281
D.5.1 Purpose 281
D.5.2 Optional Parameter 281
D.6 ENDTRPMGR (End Trap Manager) Command 284
D.6.1 Purpose 284
D.7 STRTCPSVR (Start TCP/IP Server) Command 285
D.7.1 Purpose 285
D.7.2 Optional Parameter 285
D.8 STRTRPMGR (Start Trap Manager) Command 289
D.8.1 Purpose 289
D.8.2 Optional Parameters 289
D.9 RMVCOMSNMP(Remove Community for SNMP) Command 291
D.9.1 Purpose 291
D.9.2 Required Parameter 291
D.9.3 Optional Parameter 291

Appendix E. Customizing the NetView for OS/2 Discovery Process 293
E.1 Creating a Seed File 293
E.2 Creating a Mask File 294

Appendix F. SNMPv2 297
F.1 Security 297
F.2 Operational Model 297
F.3 SNMPv2 RFCs 298

Appendix G. Special Notices 299

Appendix H. Related Publications 301
H.1 International Technical Support Organization Publications 301
H.2 Redbooks on CD-ROMs 301
H.3 Other Publications 301

How to Get ITSO Redbooks 303
How IBM Employees Can Get ITSO Redbooks 303
How Customers Can Get ITSO Redbooks 304
IBM Redbook Order Form 305

Index 307

ITSO Redbook Evaluation 309

Preface

This redbook will help anyone who needs to understand the SNMP support provided by the AS/400. As an aid to those new to SNMP, the book gives an overview of SNMP, the type of information made available via SNMP and the level of management made possible through the AS/400 SNMP agent. The book includes information on the configuration of the AS/400 SNMP support and provides examples of the use of an SNMP manager to retrieve SNMP management information from the AS/400. In showing this, the book not only shows how this can be done but also provides examples of the type of information made available to an SNMP manager when querying an AS/400 SNMP MIB. Those familiar with SNMP will find information about the standard,as well as IBM enterprise and Novell enterprise MIBs supported by the AS/400 SNMP agent.

Of special value is the information included on how the SNMP set function can be used to manage AS/400 communications objects, for example, how set can be used to re-activate a failed line description.

Where applicable, the book compares SNMP-based management with the equivalent native AS/400 function. For example, it compares the use of SNMP to display IPX routing information with an AS/400 command that can gather the same information.

The book also includes information on the SNMP manager and subagent APIs included in OS/400, how these can be used by third-party applications and how, in the case of the manager APIs, they are used by IBM-supplied OS/400 applications.

The book is valid for OS/400 releases V4R1, V3R7, V3R6, V3R2 and V3R1. It should be noted that there were no SNMP enhancements in V4R1.

Some knowledge of TCP/IP is assumed.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the Systems Management and Networking ITSO Center, Raleigh.

Mick Lugton is a Systems Engineer at the Systems Management and Networking ITSO Center, Raleigh. He writes extensively and teaches IBM classes worldwide on all areas of AS/400 cross-platform connectivity and systems management. Before joining the ITSO 3 years ago, Mick worked in the AS/400 Business Unit, UK as a Networking Specialist.

Bohuslav Endt is a Systems Engineer in IBM Czech Republic working in the Field Support Center for the AS/400 system. He has 7 years of experience with AS/400 system. His areas of expertise include the Integrated PC File Server and PC connectivity to the AS/400 system, especially Client Access, Novell NetWare and general networking.

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

John Pechacek
IBM Rochester

Ed Boden
IBM Rochester

Frank Gruber
IBM Rochester

Alan Olson
IBM Rochester

The authors of the first edition of this document were:

Maria Eugenia Hermelo
IBM Argentina

Francisco Salvador Infante de la Cruz
IBM Mexico

Lee Hargreaves
IBM UK

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 309 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

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

- Send us a note at the following address:

  `redbook@vnet.ibm.com`

Chapter 1. Introduction to SNMP

In this chapter we introduce heterogeneous networks and how SNMP has evolved to become the *de facto* method of managing these networks. We also introduce the OSI protocol stack, how the TCP/IP protocol stack aligns with this and where SNMP fits within the TCP/IP stack. We finish the chapter by discussing the SNMP definitions used later in the book.

1.1 Managing a Heterogeneous Network

Today there are many manufacturers producing hundreds of devices such as personal computers, routers and mainframes which support TCP/IP. Due to the open nature of TCP/IP and the Internet, many networks have become heterogeneous and multivendor in makeup. Vendor-specific network management tools were found to be unusable in these environments. It became obvious that an *open* network management technology was required to manage these networks. Thus, SNMP has become the industry standard network management protocol for heterogeneous networks.

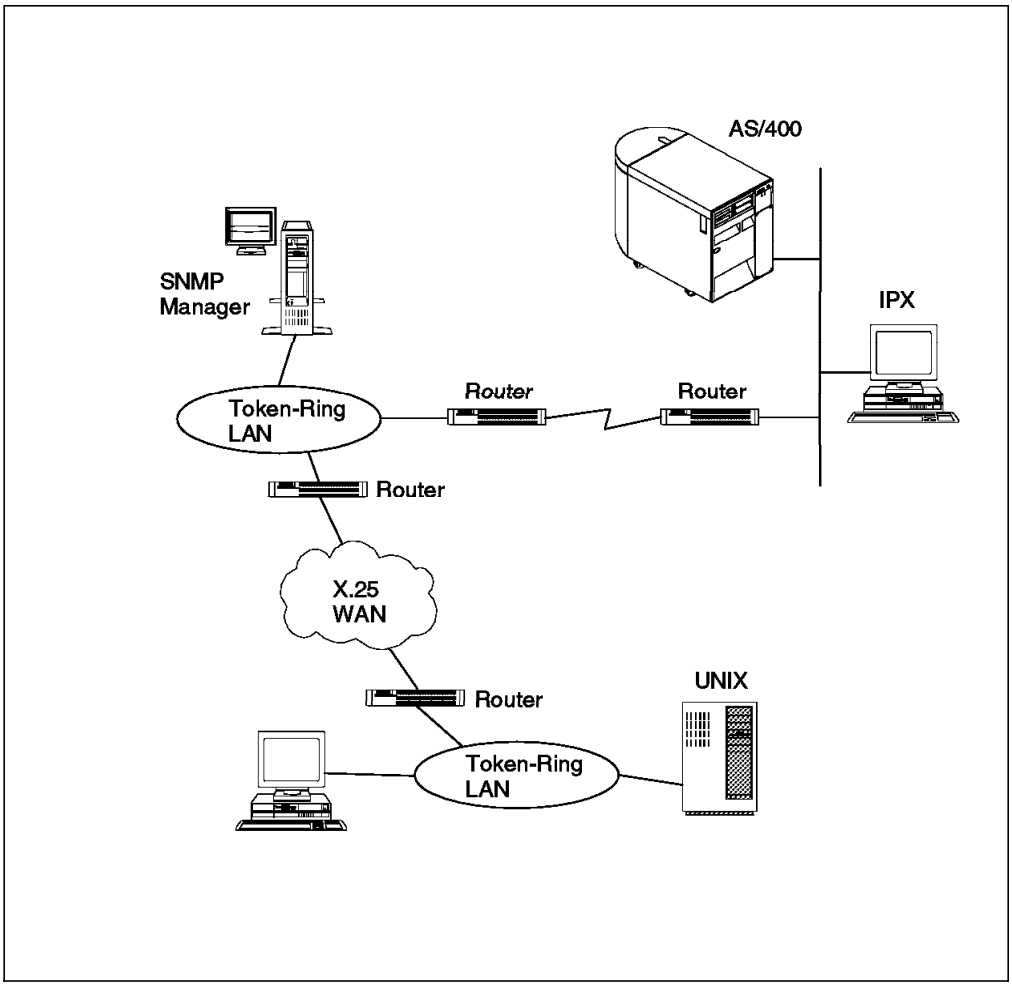


Figure 1. A Typical Heterogeneous Network

1.2 A Brief View into SNMP History

In 1968, the U.S. Defense Advanced Research Projects Agency (DARPA) began an effort to develop a technology which is now known as packet switching. This technology was strongly influenced by the development of low-cost minicomputers and digital telecommunications techniques during the 1960s. In the early 1970s, the DARPA sponsored several programs to explore the use of packet switching methods in alternative media such as mobile radio and satellite.

The expansion of the Internet drew support from U.S. government organizations including DARPA, the National Science Foundation (NSF), the Department of Energy (DOE), and the National Aeronautics and Space Administration (NASA). Eventually, international research bodies also got involved in the Internet.

Due to the successful implementation of packet radio and packet satellite technology, the desire to connect the DARPA network, ARPANET, with other packet nets arose. This led to the development of an internetwork protocol and a set of gateways to connect the different networks. DARPA sponsored further development of this solution, which resulted in a collection of computer communications protocols based on the original Transmission Control Protocol (TCP) and the lower level Internet Protocol (IP). During the course of the research, many other protocols were developed. These protocols, together with TCP and IP, are referred to as the TCP/IP Protocol Suite. A protocol suite is a set of protocols that work cooperatively together.

During these early stages, network management was of a proprietary nature due to the fact that networks were constructed with vendor-specific technology. In recognition of the need for a network management framework suitable for non-proprietary technology, in the late 1970s, the *International Organization for Standardization (ISO)*, together with the *International Telephone and Telegraph Consultative Committee (CCITT)*, started a research effort on this subject, resulting in the *Open Systems Interconnection (OSI)* protocol suite.

As the number of interconnected networks began to increase during the 1980s, the management of the Internet grew more complicated because the networks were using equipment from different vendors. In order to meet the network management demands at hand, the *Internet Activities Board (IAB)* defined a strategy formed by two parts:

- In the short term, the *Simple Gateway Monitoring Protocol (SGMP)*, being of simpler nature than the OSI model, would be modified in order to produce a new protocol for managing nodes in the Internet community.
- In the long term, the network management protocol called *Common Management Information Protocol (CMIP)*, used in the OSI model would continue to be observed.

The enhancements made to SGMP eventually originated SNMP. Currently, the Simple Network Management Protocol (SNMP) is an industry standard protocol which is used for network and system management. SNMP is a collection of specifications which describe how to manage and control a Network Element (SNMP agent) from a Network Managing Station (SNMP manager). The SNMP specifications are contained in documents called *Request for Comments (RFC)*, which are controlled by the IAB.

The RFCs that define the SNMP specifications are the following:

- RFC1155 Structure and Identification of Management Information for TCP/IP-based Internets
- RFC1212 Concise MIB Definition
- RFC1213 Management Information Base for Network Management of TCP/IP-based Internets: MIB-II
- RFC1157 Simple Network Management Protocol (SNMP)

For further details about the IAB, and RFCs, see Appendix B, “The IAB” on page 245.

Although SNMP is used predominantly in TCP/IP-based networks, AnyNet sockets over SNA allows SNMP support to be used in SNA networks.

Topics discussed in this book refer to Version 1 of SNMP (SNMPv1). Currently, Version 2 of SNMP (SNMPv2) is in the process of becoming a standard protocol. Due to the changing nature of new Internet projects, it is out of the scope of this book to give any more than a brief introduction to SNMPv2 in Appendix A, “Additional Information on SNMP” on page 235.

1.3 Protocol Layering

To help understand ideas and the way protocols work, computer communications have been divided into a stack of segments or *layers*, each responsible for carrying out specific tasks. Although this conceptual organization of protocol layers does not explain all the details involved in the communications process, the primary tasks and responsible entities at each layer are easier to identify, thus providing a comprehensive view of the overall communication processes. The TCP/IP protocol suite, and the OSI protocol suite have protocol layering models which represent them.

1.3.1 The OSI Reference Model

ISO/CCITT research originated the OSI protocol suite which is represented through the OSI reference model. The OSI model divides computer communications tasks into seven functional layers, as you can see in Figure 2 on page 4. These layers are numbered from the bottom up and are used to describe what communications operation is being done and where (regardless of the computer’s architecture).

OSI defines a *service* to be a set of functions that are offered through a *Service Access Point (SAP)*. A service is offered through the use of a *protocol*. OSI standards are grouped into pairs of service and protocol. A service may also be comprised of several simpler services from the underlying level to form a more complex service to offer to the layer above. This is known as *layering*.

The OSI layers involve two types of communications services:

Connection-Oriented The service comprises three phases summarized as *connection establishment*, *data transfer*, and *connection release*.

Connectionless The service only involves the *data transfer* phase.

While the OSI model is primarily connection-oriented, services from the lower layers do use connectionless services to provide connection oriented services to the upper layers. The upper layers are primarily connection-oriented.

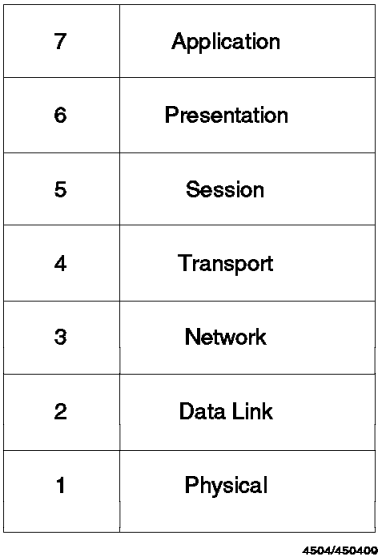


Figure 2. The OSI Reference Model

The OSI approach to network management is based on the *Common Management Information Service (CMIS)*, and its associated protocol, *Common Management Information Protocol (CMIP)*, running over a TCP connection. This is also known as *CMIP over TCP (CMOT)*.

1.3.2 Time Sequence Diagrams

Services can be represented in a time sequence diagram to show that a service is made up of one or more *primitives* and the sequence in which each service primitive occurs.

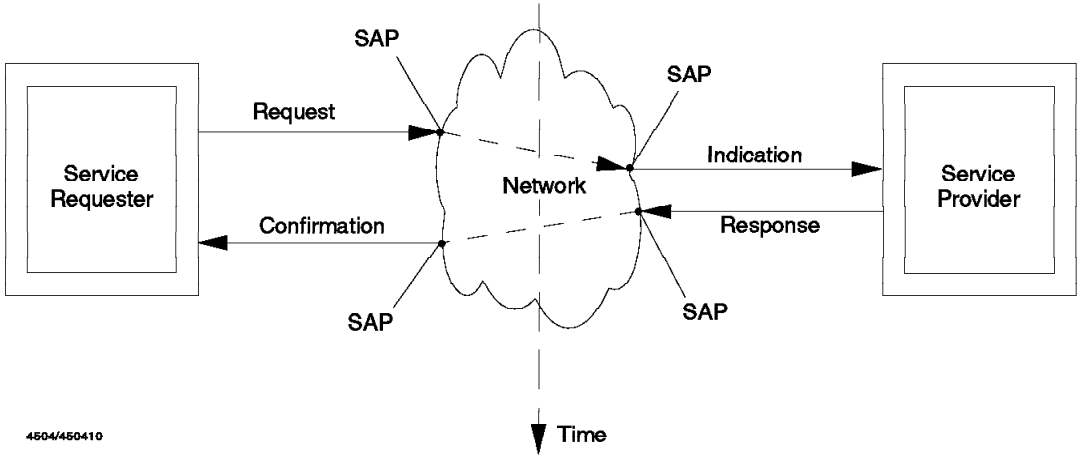


Figure 3. Service Primitives in a Time Sequence Diagram

1.3.3 The TCP/IP Protocol Suite Model

Research of the TCP/IP protocol suite originated the model for this set of protocols. This model is similar to the OSI reference model, but it is not the same. The TCP/IP protocol suite model is divided into four conceptual layers built on top of a fifth *hardware* layer, as you can see in Figure 4. A brief explanation of each layer follows:

Application Layer It is the highest level layer, where users invoke application programs that interact with services offered in a TCP/IP Internetwork.

Transport Layer It provides communication between application programs, also known as *end-to-end* services.

Internet Layer It handles communication between machines.

Network Interface Layer It is the lowest level layer, where IP datagrams are handled to be sent to a specific network.

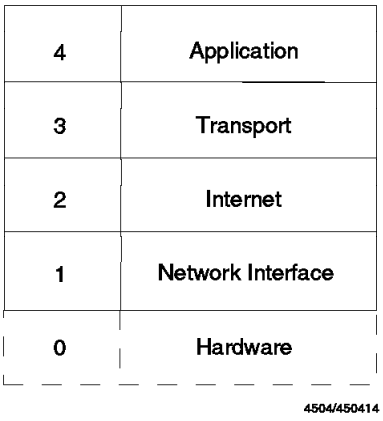


Figure 4. The TCP/IP Protocol Suite Model

Because the TCP/IP network could consist of more than just one network type, management applications operate at the application layer. For example, a TCP/IP network could consist of multiple token rings, Ethernets, and X.25 networks connected through gateways, routers, and bridges. However, network management applications that operate at the application layer, such as SNMP, are designed to operate regardless of the network hardware operating in the lower layers. An advantage of this is that networks can be managed with a limited set of protocols or commands because the different network elements would respond to the same set of commands. A disadvantage of managing at the application layer is that operating becomes dependent on a functional underlying operating system, IP software, transport protocol, and such. For example, if the TCP/IP software on a remote gateway were to fail, you would not be able to contact the managing application at that gateway because in order to reach this gateway, TCP/IP is required to be operational.

1.4 SNMP Definitions

SNMP Agent An SNMP agent is an implementation of a network management application which is resident on a managed system. Each node that is to be monitored or managed by an SNMP manager in a TCP/IP network, must have an SNMP agent resident. The agent receives requests to either retrieve or modify management information by referencing MIB objects. MIB objects are referenced by the agent whenever a valid request from an SNMP manager is received.

SNMP Manager An SNMP manager refers to a managing system that executes a managing application or suite of applications. These applications depend on MIB objects for information that resides on the managed systems.

SNMP Subagent An SNMP subagent is the implementation of a network management application on a managed system, which interfaces with the SNMP agent for the purpose of expanding the number of MIB objects that an SNMP manager can access. SNMP agents have predefined MIB objects that they can access. This limits the managing application in regards to the type of information that it can request. The need to overcome this limitation brought about the introduction of subagents. A subagent allows the dynamic addition of other MIB objects without the need to change the agent. Whether a MIB object is referenced by the agent or the subagent is transparent to the managing system.

SNMP Proxy Agent An SNMP proxy agent is one that acts on behalf of a managed system that is not reached directly by the managing system. A proxy agent is used when a managed system does not support SNMP, or when a managed system supports SNMP but for other reasons it is more convenient to manage it indirectly, for instance, through the use of a proxy agent.

Management Information Base (MIB) A Management Information Base (MIB) is a logical database residing in the managed system which defines a set of MIB objects. A MIB is considered a logical database because actual data is not stored in it, but rather provides a view of the data that can be accessed on a managed system.

MIB Object A MIB object is a unit of managed information that specifically describes an aspect of a system, for example, CPU utilization, software name, hardware type, and more. A collection of related MIB objects is defined as a Management Information Base (MIB). A MIB object is sometimes called a MIB variable.

Instance An instance refers to a particular representation of a MIB object. The MIB object which it represents can be thought of as a template for which one or more instances can be defined, depending on the type of MIB object. Actual values can only be assigned to instances of a MIB object.

SNMP Community An SNMP community is an administrative relationship between an SNMP agent and one or more SNMP managers. Each community consists of a community name, an object access specification and a list of SNMP managers′ IP addresses. A community is used by an SNMP agent to determine which requests are to be honored.

Heterogeneous Network A heterogeneous network is that in which a collection of systems of different type and manufacturer are interconnected by a variety of communication methods and protocols.

Request For Comments (RFC) A Request for Comments (RFC) is a technical report that documents standards, protocols, and guidelines for the development of TCP/IP protocol standards. RFCs are the mechanism by which TCP/IP and the Internet protocol suite are evolving. Research ideas and new protocols are documented and brought to the attention of the Internet community in the form of an RFC. Some RFCs describe protocols and applications that are so useful that they are recommended to be implemented in all future implementations of TCP/IP; that is, they become recommended protocols or *de facto* standards.

Request/Response Protocol A request/response protocol is one where in a communications environment the exchange of information among different entities is done through requests which are received by an entity for processing, after which it generates a response to be sent back to the originator of the request. SNMP uses this type of protocol to transfer data between managers and agents. The SNMP manager can send a request to the SNMP agent which will in return send a response.

SNMP Trap An SNMP trap is a message that is originated by an agent application to alert a managing application of the occurrence of an extraordinary event. SNMP traps include: coldStart, warmStart, linkDown, linkUp, authenticationFailure, EGPNeighborLoss, and enterpriseSpecific.

OBJECT IDENTIFIER (OID) An OBJECT IDENTIFIER is a means for identifying some object, regardless of the semantics associated with the object. An example would be a network object or a standards document. OBJECT IDENTIFIER is defined by ASN.1.

Chapter 2. Simple Network Management Protocol (SNMP)

SNMP is a transaction-oriented protocol that allows network elements to be queried directly. It is a simple protocol that allows management information for a network element to be inspected or altered by a system administrator at a network management station. SNMP is a TCP/IP network management protocol and is based on a manager-agent interaction. The SNMP manager (such as NetView for OS/2) communicates with its agents. Agents gather management data while the managers solicit this data and process it. An agent can also send unsolicited information, called a *trap*, to a managing system to inform it of an event that has taken place at the agent system. For example, an agent can send a trap of type of *linkDown* to the manager to inform it about the loss of a communication link with a particular device.

2.1 The SNMP Architecture

The SNMP architectural model is a collection of network management stations and network elements, such as gateways, routers, bridges and hosts. These elements act as servers and contain management agents which perform the network management functions requested by the network management stations. The network management stations act as clients; they run the management applications which monitor and control network elements. SNMP provides a means of communicating between the network management stations and the agents in the network elements to send and receive information about network resources. This information can be status information, counters, identifiers, and more.

The SNMP manager polls the agents for error and statistical data. The performance of the network will be dependent upon what the polling interval is set at. The physical and logical characteristics of network objects make up a collection of information called a Management Information Base (MIB). The individual pieces of information that comprise a MIB are called MIB objects, and they reside on the agent system. These objects can be accessed and changed by the agent at the manager's request. Unsolicited data, called traps, can also be sent from the agent to the manager under certain conditions. This is how NetView for OS/2 manages network objects. Other SNMP managers could also access these MIB objects.

2.2 Goals of the SNMP Architecture

The SNMP architecture explicitly minimizes the number and complexity of management functions realized by the management agent itself. This goal is attractive in that among other benefits, it allows for the following:

- Reduced costs in developing management agent software to support the protocol
- Few restrictions on the form and complexity of management tools
- Simplified, easier to implement management functions

A second goal of the protocol is that the functionality can be extended to accommodate additional, possibly unanticipated, aspects of network management. A third goal is that the architecture be, as much as possible, independent of the architecture and mechanisms of particular hosts or gateways.

2.3 SNMP Model

The SNMP model is made up of the following components:

- At least one network element to be managed (agent system) containing an agent
- At least one network managing station (NMS), containing one or more network management applications
- A network management protocol for use by the NMS and the agent system to exchange network management information
- At least one MIB defining the information to be managed on the agent system

Figure 5 is a graphical representation of the SNMP model.

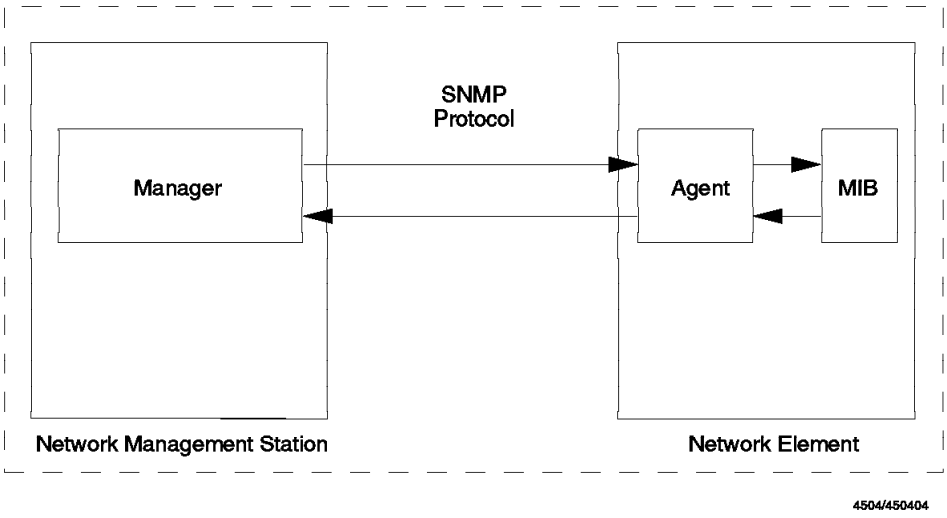


Figure 5. The SNMP Model

2.4 User Datagram Protocol (UDP)

The communication of management information among management entities is done in SNMP through the exchange of protocol messages, each of which is entirely and independently represented within a single UDP datagram using the Basic Encoding Rules (BER) of ASN.1. These protocol messages are referred to as *Protocol Data Units (PDU)*. See Appendix A, “Additional Information on SNMP” on page 235 for more details.

Consistent with the goal of minimizing complexity of the management agent, the exchange of SNMP messages requires a simple datagram service. For this reason, the preferred transport service for SNMP is the User Datagram Protocol (UDP), although the mechanisms of SNMP are generally suitable for use with a wide variety of transport services.

As a transport layer protocol, UDP uses the Internet Protocol (IP) as the underlying protocol. Two inherent characteristics of UDP provide for its simplicity. One of them is that UDP is *unreliable*, meaning that the UDP does not

guarantee that messages will not be lost, duplicated, delayed, or sent in a different order. UDP is also a *connectionless* protocol, because the only process involved is the transfer of data (see 1.3, “Protocol Layering” on page 3). However, UDP does provide a certain level of data integrity validation through checksum operations. UDP also provides application layer addressing because it has the ability to route messages to multiple destinations within a given host. Figure 6 shows where SNMP and UDP operate within the TCP/IP protocol stack.

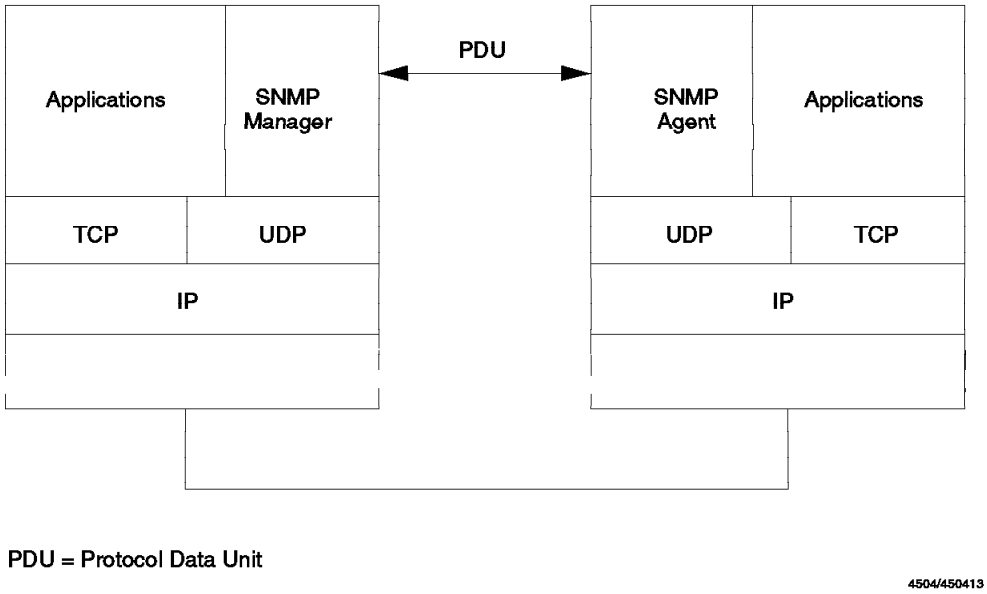


Figure 6. SNMP in the TCP/IP Protocol Stack

2.5 Asynchronous Request/Response Protocol

Managing systems generate SNMP requests, and agent systems generate responses to these requests. After a request message has been sent, SNMP does not need to wait for a response. SNMP can send other messages or realize other activities. These attributes make SNMP an asynchronous request/response protocol.

An agent system can also generate SNMP messages called *traps* without a prior request from the managing system. The purpose of a trap message is to inform the managing system of an extraordinary event that has occurred at the agent system. It must be noted that all request/response transactions are subject to the time delays inherent to all networks. The typical SNMP request/response primitives take place in the following manner:

- Manager polls agent with a REQUEST for information.
- Agent supplies information, which is defined in a MIB, in the form of a RESPONSE.

Figure 7 on page 12 illustrates two time sequence diagrams. The top diagram shows a typical SNMP request/response interaction, while the bottom diagram shows a typical SNMP trap sequence.

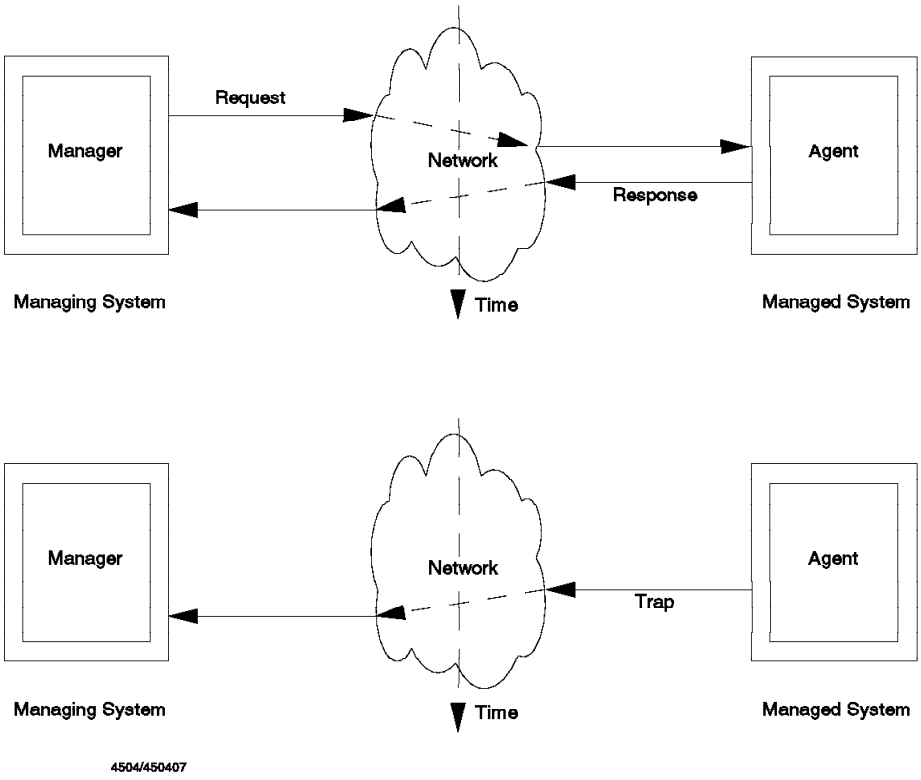


Figure 7. Asynchronous Request/Response Protocol

2.6 SNMP Agent

The SNMP agent has the following two responsibilities:

1. To gather error and statistical data defined by MIB objects
2. To react to changes in certain MIB variables made by a managing application

In summary, the following steps describe the interactions that take place in an SNMP-managed network:

- The SNMP agent gathers vital information about its respective device and networks.
- The SNMP manager polls each agent for MIB information and can display this information at the SNMP manager station. In this manner, a network administrator can manage the network from a management station.
- An agent also has the ability to send unsolicited data to the SNMP manager in the form of a trap. A trap is generally a network condition detected by an SNMP agent that requires immediate attention by the network administrator.

2.7 SNMP Subagent

A subagent extends the set of MIB objects provided by an SNMP agent. With a subagent it is possible to define MIB variables that are useful and specific to a particular environment, then register them with the SNMP agent.

Requests for the variable(s) that are received by the SNMP agent are passed to the process acting as a subagent. The subagent then returns an appropriate answer to the SNMP agent. The SNMP agent eventually sends an SNMP response with the answer back to the network managing station that initiated the request. The network management station has no knowledge that the SNMP agent calls on other processes to obtain an answer. From the viewpoint of the managing application, the agent is the only network management application on the managed system.

2.8 SNMP Manager

An SNMP manager refers to a network management station which runs a network management protocol and network management applications. SNMP is the network management protocol which provides the mechanism for management. Several different network management applications exist that can be used, such as NetView for OS/2, and NetView for AIX. The network management application provides the policy to be used for management.

The network management applications rely on Management Information Base (MIB) objects for information regarding the managed system, also called the agent system. Management systems generate requests for this MIB information and an SNMP agent on the managed system responds to these requests. A request can either be the retrieval or modification of a MIB variable.

The agent system makes network and system information available to other systems by accessing the MIB objects and allowing configuration, performance, and problem management data to be managed by the SNMP manager.

For example, a network manager can access the system description of a particular agent system by using the network management application to gain access to the agent system′s *sysDescr* MIB object. To do this, the managing application builds a message that requests a MIB object called *sysDescr*. This request is sent to the agent system where the agent decodes the message and then retrieves the information related to the *sysDescr* MIB object. The agent constructs a response with this information and sends it back to the managing application. When the application has decoded the response, the SNMP manager can then display the agent system′s description information to the user. Figure 8 on page 14 shows the relationships among the SNMP entities as discussed in the previous paragraphs.

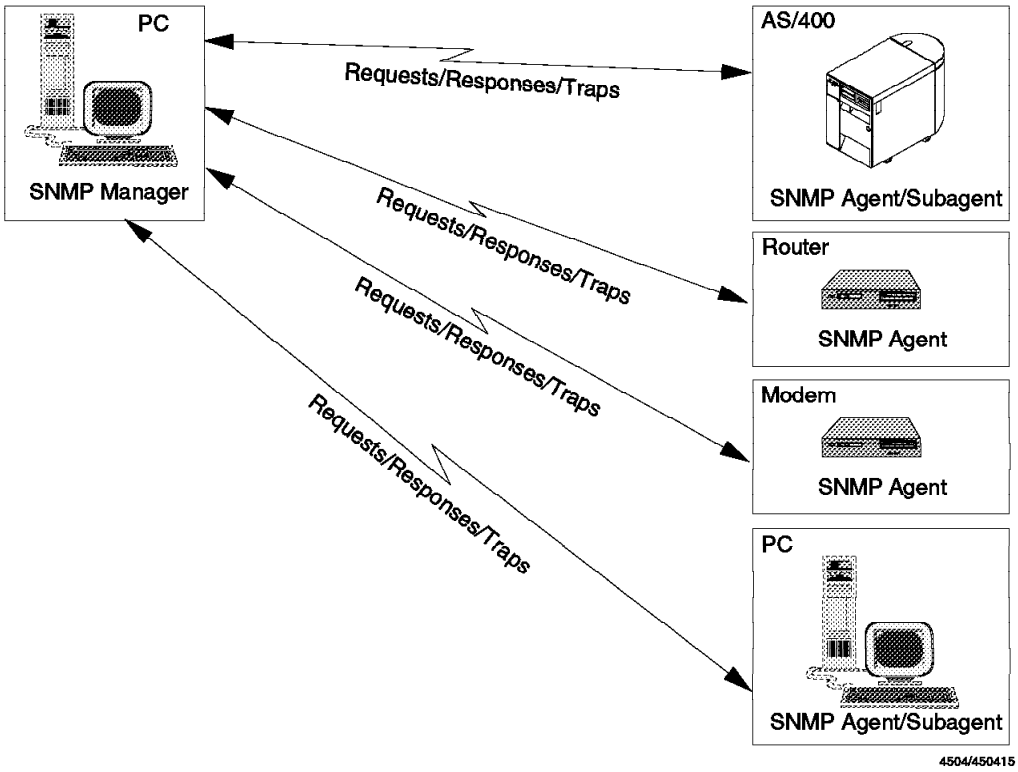


Figure 8. SNMP Manager/Agent/Subagent Relationship

2.9 Understanding MIBs

The physical and logical characteristics of a system make up a collection of information which can be managed through SNMP. The individual pieces of information make up MIB objects. A Management Information Base (MIB) is comprised of MIB objects that reside on the agent system, where they can be accessed and changed by the agent at the manager's request.

The administrative policy established by the IAB, results in a classification of MIBs according to their applicability and purpose. Therefore, MIBs are classified as follows:

Standard MIB All devices that support SNMP are also required to support a standard set of common managed object definitions of which a MIB is composed. The standard MIB object definition, MIB-II, enables you to monitor and control SNMP managed devices.

Enteprise-specific MIB SNMP permits vendors to define MIB extensions or enterprise-specific MIBs, specifically for controlling their products. An enterprise-specific MIB must follow certain definition standards just as other MIBs must, to ensure that the information they contain can be accessed and modified by SNMP agents.

Experimental MIB Generally, new ideas and activities related to the Internet result in the definition of MIB objects. An experimental MIB is comprised of such objects. This approach offers the advantage that

all new ideas must be proven while under experiment before they can be proposed for standardization.

2.9.1 Representation of Management Information

Since SNMP is used to manage a broad range of MIB objects, each and every one of these needs to be uniquely identified in order to provide unambiguous management capabilities. The following sections provide brief discussions on the guiding mechanisms through which MIB objects are uniquely identified for management purposes and by which MIBs are structured.

2.9.1.1 Abstract Syntax Notation.1 (ASN.1)

ASN.1 is a formal language originated by the ISO which is used to define information exchanged by protocols, in the form of an *abstract syntax*, meaning that data is defined without regard to a specific machine architecture. ASN.1 is very useful because it doesn't allow any ambiguities in the information it represents. ASN.1 is used to define managed objects, as well as the PDUs exchanged by the protocols that manage those objects. ASN.1 provides two representations of the information defined by it. One representation can be read by humans, and the other representation is an encoded version of the same information, which is used by the communications protocols.

Each managed object is described using an ASN.1 notation called OBJECT-TYPE. An OBJECT-TYPE definition consists of five fields represented in the following example which describes a MIB object called *hrSystemUptime* (from the Host group in the MIB-II):

```
hrSystemUptime OBJECT-TYPE
    SYNTAX TimeTicks
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
           "The amount of time since this host was last
           initialized.  Note that this is different from
           sysUpTime in MIB-II (RFC 1213) because sysUpTime is
           the uptime of the network management portion of the
           system."
    ::= { hrSystem 1 }
```

The following is a description of each of the fields that define an OBJECT-TYPE:

OBJECT DESCRIPTOR (Name) It is a textual name for the OBJECT-TYPE. For example, hrSystemUptime, and sysUpTime are OBJECT DESCRIPTORS or names for an OBJECT-TYPE.

SYNTAX It defines the data type associated with the OBJECT-TYPE. ASN.1 constructs are used to define this structure, although the full generality of ASN.1 is not permitted.

The ASN.1 type *ObjectSyntax* defines three categories of object syntax: *simple*, *application-wide*, and *simply constructed*. INTEGER, OCTET STRING, NetworkAddress, Counter, Gauge, TimeTicks, SEQUENCE, and SEQUENCE OF are all examples of these types of syntax.

ACCESS It defines the level of access permitted for the managed object. It can be one of the following levels:

- **read-only** Object instances may only be read, not set.

- **read-write** Object instances may be read, or set.
- **write-only** Object instances may only be set, not read.
- **not-accessible** Object instances may not be read or set.

STATUS It defines the managed object's implementation requirement in terms of one of the following statuses:

- **mandatory** A managed node *must* implement this object.
- **optional** A managed node may implement this object.
- **obsolete** A managed node need not implement this object anymore.

DESCRIPTION It is a textual description of the semantics of the OBJECT-TYPE. In the case of non-enterprise-specific MIB implementations, care must be taken to ensure that instances of the managed object fulfills its description since the MIB is intended for use in multivendor environments. As such it is vital that objects have consistent meanings across all machines.

2.9.1.2 Structure of Management Information (SMI)

SGMP adopted the convention of using a well-defined subset of the ASN.1 language. SNMP continues and extends this tradition by utilizing a moderately more complex subset of ASN.1 for describing managed objects and for describing the Protocol Data Units (PDUs) used for managing those objects. In addition, the desire to ease eventual transition to OSI-based network management protocols led to the definition in the ASN.1 language of an Internet standard Structure of Management Information (SMI) and Management Information Base (MIB). For the sake of simplicity, SNMP uses only a subset of the Basic Encoding Rules of ASN.1.

The SMI is defined via ASN.1 and comprises a set of rules used to describe management information in the form of MIB objects. The Internet standard RFC1155 documents the SMI in detail. Based on the SMI, management objects can be uniquely identified through the use of an OBJECT IDENTIFIER which represents a specific object's name, that is, an OBJECT DESCRIPTOR. An OBJECT IDENTIFIER can be used for purposes other than naming managed object types. For example, each international standard has an OBJECT IDENTIFIER assigned to it for the purpose of identification. In short, OBJECT IDENTIFIERs are a means for identifying some object, regardless of the semantics associated with the object. An example would be a network object or a standards document.

2.9.2 MIB Naming Conventions

MIB objects are logically organized in a hierarchy called a tree structure. Each MIB object has a label derived from its location in the tree structure. A label is a pairing of a brief textual description and an integer. An OBJECT IDENTIFIER is a sequence of non-negative integers which traverse a global tree for the purpose of identifying an object. The tree consists of a root which branches to connect to a number of labeled *nodes*, also called *subordinates*. Each node may, in turn, have children of its own which are labeled. In this case, we may term the node a *subtree*, or *intermediate node*. If a node does not have children, it is called a *leaf node*.

A fully qualified OBJECT IDENTIFIER for a particular MIB object contains all nodes, starting at the root and traversing the tree to an arbitrary level of depth, until the desired leaf object is reached. The nodes are concatenated and separated by periods, in a format known as ASN.1 notation. For example, the mib-2 subtree is iso.org.dod.internet.mgmt.mib-2, which is concisely written in ASN.1 notation as 1.3.6.1.2.1.

Note

Do not confuse the ASN.1 notation used by OBJECT IDENTIFIERs with the format used to identify IP addresses. Although both notations use periods in their format, they are not the same thing.

The OBJECT IDENTIFIER for the *sysContact* object contained in the MIB-II is 1.3.6.1.2.1.1.4, as you can see (in Figure 9) by following the labels from the root down to the leaf object.

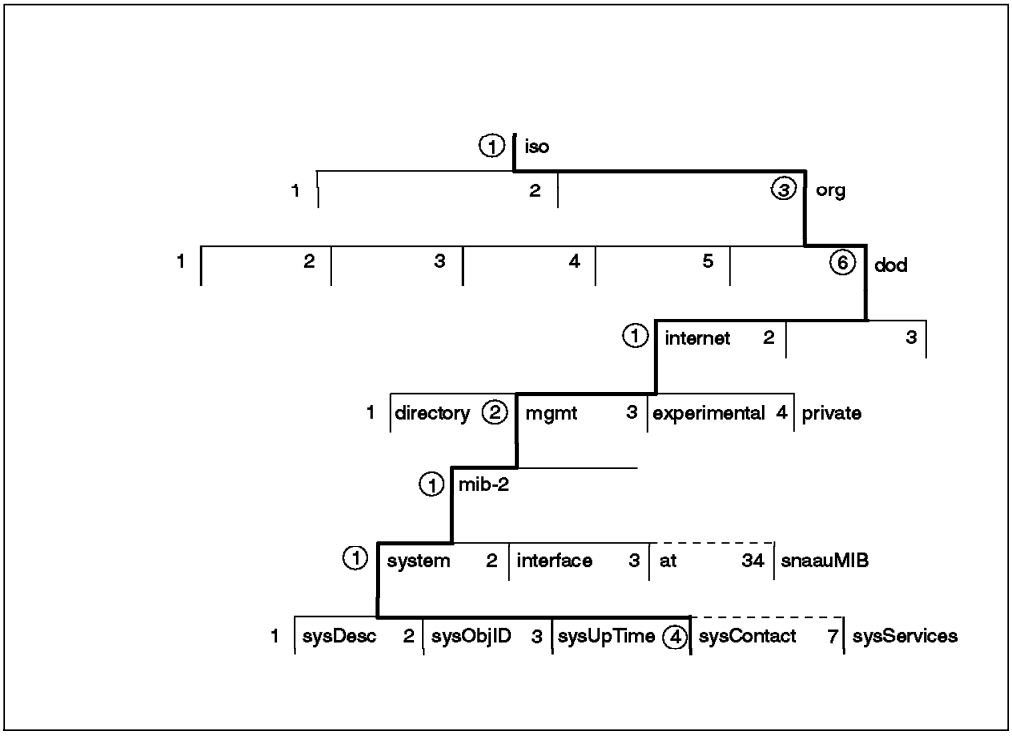


Figure 9. A View of the MIB Tree Structure

The standard MIB-II is registered in the mib-2(1) subtree. Experimental MIBs are registered in the experimental(3) subtree. Enterprise-specific MIBs are registered in the private(4) subtree. Each enterprise is assigned a number. IBM is assigned the enterprise number 2, thus all IBM enterprise-specific MIB objects have an OBJECT IDENTIFIER starting with 1.3.6.1.4.1.2, corresponding to the tree structure iso.org.dod.internet.private.enterprises.ibm. Figure 10 on page 18 shows where IBM objects reside within the global tree structure.

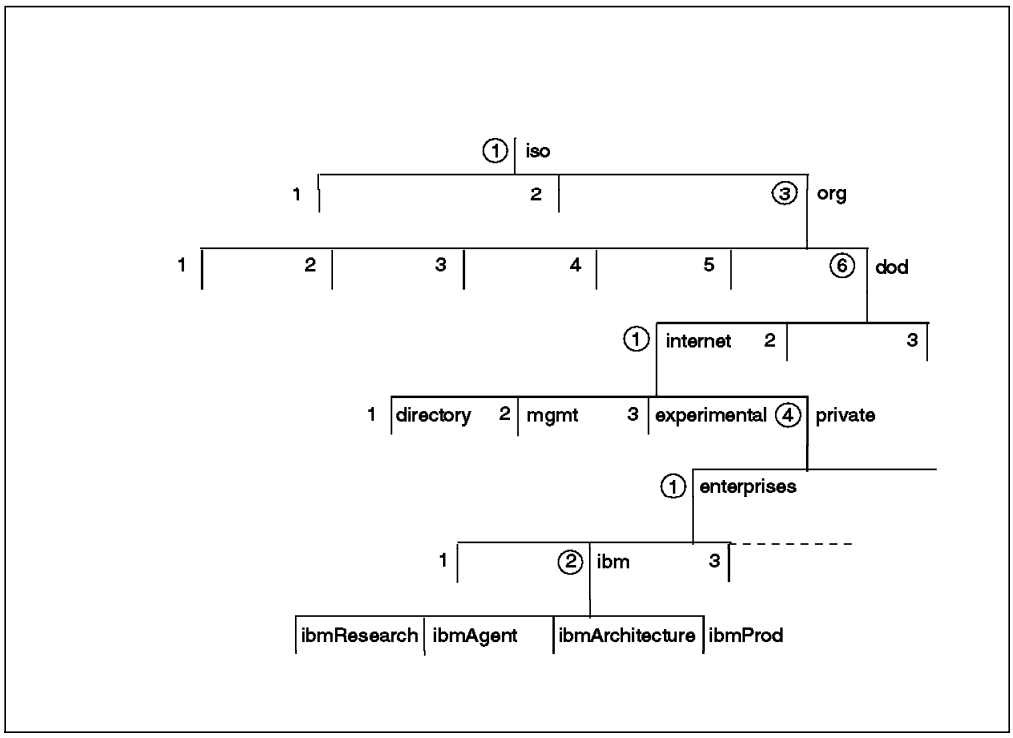


Figure 10. The MIB Tree Structure for IBM Objects

For a more detailed explanation on the MIB tree structure, see Appendix A, "Additional Information on SNMP" on page 235.

2.9.2.1 Identification of Object Instances

The names for all OBJECT-TYPEs in the MIB are defined explicitly either in the Internet standard MIB, or in other documents which conform to the naming conventions of the SMI. The SMI requires that conformant management protocols define mechanisms for identifying individual *instances* of those OBJECT-TYPEs for a particular network element. It is these instances of MIB objects that are actually managed by a management protocol, thus it is important to understand what an instance is.

To understand what an *instance* of a MIB object is, it is helpful to view a MIB object as an abstract definition, or template. A MIB object instance derives from the object's template; thus, it retains all of the object's properties, but also has a particular value assigned to it which conforms to the MIB object's OBJECT-TYPE definitions. In a way, an instance can be viewed as a snapshot of a particular value of a MIB object.

Each instance of an OBJECT-TYPE defined in the MIB is identified in SNMP operations by a unique name called its *variable name*. That is, the term *variable* refers to an instance of a managed object. In general, the name of an SNMP variable is an OBJECT IDENTIFIER of the following form:

```
x.y
```

Where *x* is the name of an OBJECT-TYPE defined in the MIB, and *y* is an OBJECT IDENTIFIER suffix that, in a way specific to the named OBJECT-TYPE, identifies the desired instance.

The rules for determining the value of the suffix (*y* in the previous example) depend on the type of object. Basically, two types of objects are involved in a

MIB. The following is an explanation of each, and the manner in which the suffix value is assigned.

Tabular Objects These objects form part of a grouping in the form of a table. Tables are useful for managing multiple variables that share common properties but may have different values, within a given device.

For example, a personal computer usually has several different software entities installed on it. The *hrSWInstalledTable* object in the Host Resources MIB (RFC1514) defines an object called *hrSWInstalledName* which is used to obtain a textual description of the software which is installed on a given system. hrSWInstalledName is a tabular object because it is part of a table, hrSWInstalledTable, which is also a tabular object itself. The different objects that make up the hrSWInstalledTable, such as hrSWInstalledIndex, hrSWInstalledName, and others, make up the *columns* of that table. Through the use of the values defined by hrSWInstalledIndex, which assigns a unique index number to each piece of software that is installed on the personal computer, it is possible to identify the instances of hrSWInstalledName. The instances of the table objects make up the *rows* of the table. Figure 11 represents this in a graphical form.

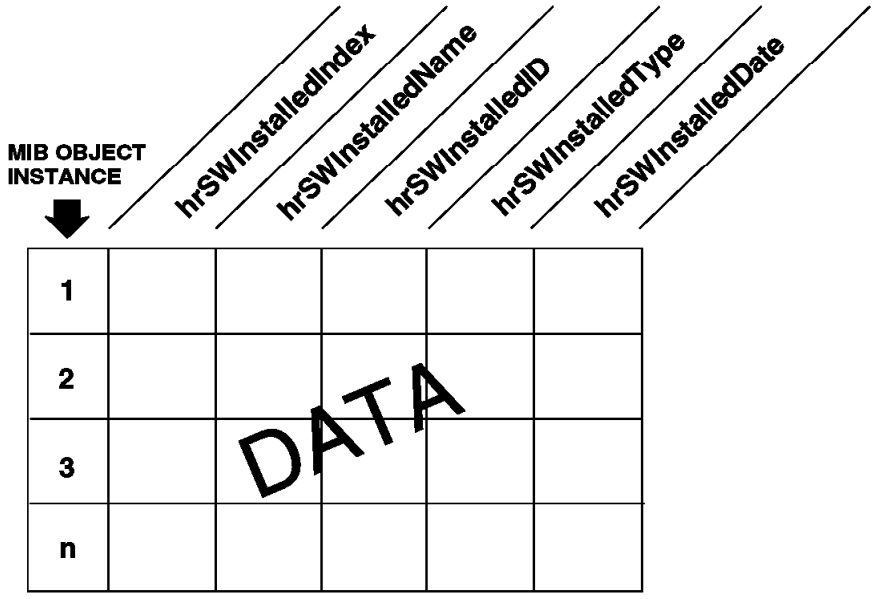


Figure 11. A Graphical Representation of the hrSWInstalledTable

In this manner, it is possible to uniquely identify each software entity that is installed on the personal computer because each one is identified by the same object, but by a different instance. Each

instance then is the suffix for the OBJECT IDENTIFIER of each piece of software. To show this, the instance of hrSWInstalledName that is associated with the first software entity is:

```
hrSWInstalledName.1      or      1.3.6.1.2.1.25.6.3.1.2.1
```

The suffix then is the same as the instance. In this example it is *1*.

By using OBJECT IDENTIFIERs to identify the different instances, it is possible to create a *lexicographic ordering* over all the object variables. This naming strategy admits the fullest exploitation of the SNMP GETNEXT operation (see 2.9.1.2, “Structure of Management Information (SMI)” on page 16) because it assigns names for related variables so as to be contiguous in the lexicographical ordering of all variable names known in the MIB.

Scalar (non-tabular) Objects These objects do not form part of a table, and there is only one instance of these objects in a given device. Thus, the only OBJECT IDENTIFIER suffix, *y*, for the OBJECT-TYPE, is zero. For example, the OBJECT IDENTIFIER for OBJECT-TYPE sysUpTime is simply:

```
sysUpTime.0     or     1.3.6.1.2.1.1.3.0
```

2.10 SNMP Operations

To be consistent with its simplicity objective, SNMP contains few commands to execute its operations. SNMP supports two commands that managing systems can use to retrieve information from a managed system and one command to store a value into a managed system. All other operations are considered to be side-effects of these three commands.

As an example, SNMP does not contain an explicit *reboot* command. However, this action might be invoked by simply setting a parameter indicating the number of seconds until system reboot. In addition to these commands, SNMP supports an event-driven mechanism used to alert managing stations of the occurrence of extraordinary events at a managed system.

The approach that SNMP is based on for network management makes it a simple, stable, and flexible protocol because it can accommodate new operations as side-effects of the same SNMP commands acting upon new MIB variables; thus not requiring SNMP to be modified.

SNMP also specifies that if a single SNMP message specifies operations on multiple variables, the operations will either be performed on all variables or on none of them. No operation will be performed if any of the variables are in error.

Each SNMP operation is defined in a particular PDU, a brief description of each operation follows (see A.3.2, “SNMP Protocol Data Units” on page 241 for more details).

- **GET**. This is a request originated by a managing application to retrieve an instance of one or more MIB objects. The specified instance is retrieved for each variable in the request, provided that community profile authentication has been successful.

- **GETNEXT**. This is a request originated by a managing application to retrieve the next valid instance following the specified instance of one or more MIB objects, provided that community profile authentication has been successful.
- **SET**. This is a request originated by a managing application to store a specific value for one or more MIB variables. All variables must be updated simultaneously, or none of them.
- **GET-RESPONSE**. This is response data that is originated by an agent application and is sent back to the originator of a GET, GETNEXT, or SET request.
- **TRAP**. This is an unsolicited message originated by an agent application which is sent to one or more managing systems within the correct community, to alert them of the occurrence of an event. Traps include the following types:
  - coldStart (0)
  - warmStart (1)
  - linkDown (2)
  - linkUp (3)
  - authenticationFailure (4)
  - egpNeighborLoss (5)
  - enterpriseSpecific (6)

2.11 Table Traversal

An important use of the GETNEXT operation is the traversal of conceptual tables of information within the MIB. With table traversal it is possible to view all MIB objects as if they were organized in a table. The following example may help to understand this.

We will assume that an SNMP application has extracted the table index and software type for each entry in the installed software table (hrSWInstalledTable) of a particular network element. Suppose that this installed software table has the following three entries:

```
hrSWInstalledIndex            hrSWInstalledType

1.3.6.1.2.1.25.6.3.1.1.3      4 (application)
1.3.6.1.2.1.25.6.3.1.1.1      2 (operatingSystem)
1.3.6.1.2.1.25.6.3.1.1.2      1 (unknown)
```

The management station sends to the SNMP agent a GETNEXT request containing the indicated OBJECT IDENTIFIER values as the requested variable names:

```
PDU ==>  GetNextRequest ( hrSWInstalledIndex, hrSWInstalledType )
```

The SNMP agent's GET-RESPONSE will contain:

```
PDU ==>  GetResponse (( hrSWInstalledIndex.1 = 1 ),
                      ( hrSWInstalledType.1 = 2 ))
```

The management station continues with:

```
GetNextRequest ( hrSWInstalledIndex.1, hrSWInstalledType.1 )
```

The SNMP agent responds:

```
GetResponse (( hrSWInstalledIndex.2 = 2 )
             ( hrSWInstalledType.2 = 1 ))
```

The management station continues with:

```
GetNextRequest (hrSWInstalledIndex.2, hrSWInstalledType.2 )
```

The SNMP agent responds:

```
GetResponse (( hrSWInstalledIndex.3 = 3 )
             ( hrSWInstalledType.3 = 4 ))
```

The management station continues with:

```
GetNextRequest (hrSWInstalledIndex.3, hrSWInstalledType.3 )
```

As there are no further entries in the table, the SNMP agent returns those objects that are next in the lexicographical ordering of the known object names. This response signals the end of the routing table to the management station.

Chapter 3. OS/400 SNMP Support

AS/400 support for SNMP was provided at OS/400 Version 3 Release 2 and Version 3 Release 6 (limited support was included within OS/400 V3R1).

OS/400 SNMP support is comprised of an SNMP Agent and APIs. These APIs can be used to build SNMP management and SNMP subagent applications.

When using OS/400 AnyNet support, SNMP can also be used over an SNA network. AnyNet is also a part of OS/400 Version 3 Release 2 and Version 3 Release 6. For a more detailed explanation refer to Chapter 14, “SNMP and AnyNet” on page 227.

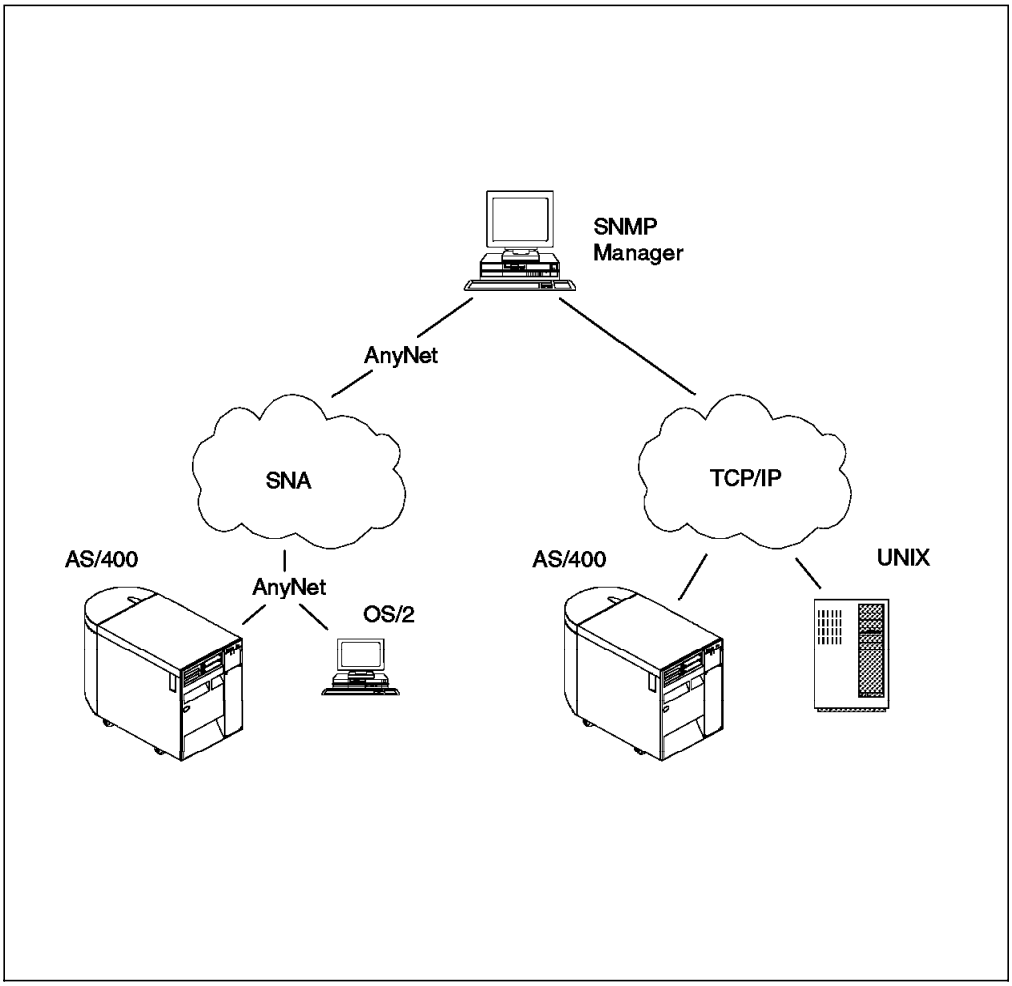


Figure 12. OS/400 SNMP Support over TCP/IP and SNA Network.

The OS/400 SNMP implementations are based on the following Internet RFCs:

- RFC1157 Simple Network Management Protocol (SNMP)
- RFC1155 Structure and Identification of Management Information for TCP/IP-based Internet.
- RFC1212 Concise MIB Definition
- RFC1213 Management Information Base for Network Management of TCP/IP-based Internet: MIB-II

- RFC1215 Convention for Defining Traps for use with SNMP
- RFC1280 IAB Official Protocol Standards
- RFC1340 Assigned Numbers

3.1 OS/400 SNMP Agent

The OS/400 SNMP agent support became a feature of base OS/400 with versions V3R2 and V3R6. This support allows the AS/400 to be managed by a remote host. For example:

- An RS/6000 running AIX and NetView for AIX
- A Personal Computer running NetView for OS/2

The OS/400 SNMP agent support allows configuration, performance and problem management data to be available to SNMP managers. Typically, the SNMP agent will keep statistics on the status of its network interfaces, incoming and outgoing traffic, dropped datagrams and error messages generated. This information will be requested by an SNMP manager. These units of managed information that specifically describe an aspect of a system such as system name, incoming traffic, hardware number, are MIB objects. A collection of related MIB objects is defined as a MIB. The OS/400 SNMP agent supports the following standard RFC MIBs, IBM enterprise MIBs and Experimental MIB (for more information on the different types of MIBs refer to 2.9, “Understanding MIBs” on page 14). For a complete list of the MIBs supported, see Chapter 9, “OS/400 Supported MIBs” on page 147.

3.1.1 Standard RFC MIBs

The following standard MIBs were supported at OS/400 V3R2 and V3R6:

MIB-II (RFC1213) MIB-II describes those objects that are implemented by managed nodes running the Internet suite of protocols.

The AS/400 implements 9 of the 10 MIB-II groups. The EGP (Exterior Gateway Protocol) group is not supported by OS/400 SNMP - the AS/400 does not provide Exterior Gateway Protocol support.

The following are examples of some of the MIB objects supported:

sysContact This MIB-II object belongs to the system group. It provides the identification of the contact person for this managed node; it can also contain information on how to contact this person.

ipInReceives This MIB-II object belongs to the IP group. It provides the total number of input datagrams received, including those received in error.

tcpInSegs This MIB-II object belongs to the TCP group. It provides the total number of segments received, including those received in error.

Ethernet-like (RFC1398) The Ethernet-like interface MIB defines objects for managing Ethernet-like object interface types. The Ethernet-like MIB belongs to the transmission group of MIB-II.

The following are examples of some of the MIB objects supported:

dot3StatsDeferredTransmissions This Ethernet-like MIB object belongs to the dot3StatsTable group. It provides a count of frames for which the first transmission attempt on a particular interface was delayed because the medium was busy.

dot3StatsFrameTooLongs This Ethernet-like MIB object belongs to the dot3StatsTable group. It provides a count of frames received on a particular interface that exceeded the maximun permitted size.

FDDI (RFC1285) The FDDI MIB defines objects for managing devices which implement FDDI. The FDDI MIB belongs to the transmission group of MIB-II.

The following are examples of some of the MIB objects supported:

snmpFddiSMTRemoteDisconnectFlag This FDDI MIB object belongs to the snmpFddiSMT group. It indicates that the station was remotely disconnected from the network.

snmpFddiPORTLemRejectCts This FDDI MIB object belongs to the snmpFddiPORT group. It provides a count indicating the times a link was rejected.

Frame Relay (RFC1315) This MIB defines frame relay objects for managing frame relay. The frame relay MIB belongs to the transmission group of MIB-II.

The following are examples of some of the MIB objects supported:

frDlcmiAddress This frame relay MIB object belongs to the frDlcmiTable group. It indicates which address format is in use on the frame relay interface.

frCircuitState This frame relay MIB object belongs to the frCircuitTable group. It indicates whether a particular virtual circuit is operational.

Token-ring (RFC1231) The IEEE 802.5 token-ring MIB defines managed objects used for managing subnetworks which use the IEEE 802.5 token-ring technology described in the 802.5 token-ring Access Method and Physical Layer Specification, IEEE standard 802.5-1989. The token-ring MIB belongs to the transmission group of MIB-II.

The following are examples of some of the MIB objects supported:

dot5RingStatus This token-ring MIB object belongs to the dot5Table group. It provides the current interface status.

dot5RingOpenStatus This token-ring MIB object belongs to the dot5Table group. It indicates the success, or the reason for failure, of the station's most recent attempt to enter the ring.

Host (RFC1514) This MIB contains host related data such as what hardware and software is installed. We look at this MIB in greater detail in Chapter 10, "Host Resources MIB" on page 189.

3.1.2 IBM Enterprise MIBs

The following IBM enterprise MIBs were supported at OS/400 V3R2 and V3R6:

APPN MIB The APPN Node Group provides global information about the APPN node, which is either a network node or an end node.

The APPN Topology Group represents the entire APPN network topology including network nodes, virtual nodes and all transmission groups associated with these nodes.

The APPN Local Topology Group describes the local topology. This MIB group defines the required objects for retrieval of information about this node and the objects that represent the local topology about end nodes. The APPN MIB was enhanced at OS/400 V3R6 to include the management of APPC communications objects (lines and controllers).

Client Management MIB The Client System Group provides global information about the client connectivity and capabilities.

The Client Hardware Group provides information related to storage, devices, file systems and printers.

The Client Software Group provides information related to installed software and fixes.

A detailed description of this MIB is provided in Chapter 8, “OS/400 SNMP-Based Functions” on page 127.

NetView/6000 Subagent MIB One MIB object in this MIB is supported. The MIB object is *nv6saComputerSystemLoad* belonging to the group nv6saComputerSystem. The object provides the average percentage of load (processor utilization) during the elapsed time. Each retrieval of this value is calculated in the same manner as the value displayed in the DSPSYSACT command when the *restart* function key is pressed. Refer to 5.1.5, “The NetView for OS/2 Grapher Function” on page 57 for an example of this MIB.

Remote Workstation MIB The Remote Workstation MIB provides information about 5494 remote workstation controllers attached to the AS/400 and workstations attached to these controllers.

A detailed description of this MIB is provided in 9.4.4, “AS/400 Remote Workstation MIB” on page 171.

3.1.3 Novell Enterprise MIBs

OS/400 V3R2 and V3R7 added support for Novell networking. As part of this enhancement, support for the following MIBs was also added:

Internetwork Packet Exchange MIB
This MIB provides information about IPX environment on the AS/400.

Routing Information and Service Advertising Protocols MIB
This MIB provides information about routing and services supplied by the IPX network.

Netware Link Services Protocol MIB
This MIB provides information about routing and services information supplied via the IPX NLSP protocol.

We will look in more detail at Novell MIBs in 9.5, “Novell Enterprise MIBs” on page 178.

3.1.4 DPI 2.0

The DPI 2.0 MIB defines an extension to SNMP agents that permits end users to dynamically add, delete, or replace management variables in the local MIB without requiring recompilation of the SNMP agent.

3.1.5 SNMP Subagent MIB

The SNMP subagent MIB provides information about subagents to facilitate management activities. The MIB is designed to handle multiple types of subagents, including DPI subagents (see RFC1592), for which the DPI 2.0 API is provided. All the SA MIB information is dynamic, for the duration of the SNMP agent job. The SA MIB consists of six summary OIDs and two tables For more details refer to 9.3, “SNMP Subagent MIB” on page 158.

3.1.6 Trap Support

A trap is an unsolicted message originated by an SNMP agent. In addition to being able to generate a trap, OS/400 also has the ability to forward a received trap to another SNMP manager. The OS/400 SNMP agent can generate the following different types of traps:

coldStart

warmStart

linkDown

linkUp

authenticationFailure

enterpriseSpecific

Refer to A.3.2.5, “The Trap-PDU” on page 243 for a description of the different traps.

For more detailed information on configuring and using the AS/400 trap support, see Chapter 11, “AS/400 as a Trap Generator” on page 205.

3.2 OS/400 SNMP Manager

Full SNMP manager support is not currently implemented by the AS/400 but a set of APIs is provided to help build an SNMP manager. An OS/400 SNMP manager can be designed using both the SNMP manager APIs and the SNMP trap support.

The SNMP manager APIs provide SNMP management applications the ability to perform SNMP management functions (GET, GETNEXT, and SET) to local or remote SNMP agents. Refer to 6.1, “SNMP Manager APIs Overview” on page 75 for more information on the manager APIs.

The trap manager receives traps, parses traps, and then enqueues the traps to an internal queue. The OS/400 SNMP agent also has the ability to forward traps to other managers. Trap forwarding is configurable using a CL command interface. Refer to 6.2, “Trap Manager Overview” on page 78 for more information on trap forwarding.

3.3 OS/400 SNMP Subagent

OS/400 Version 3 Release 2 and Version 3 Release 6 added support for an SNMP subagent API. Programs using the API must be written in the C language. The SNMP agent Distributed Protocol Interface (DPI) permits users to dynamically add, delete, or replace management variables in the local MIB without requiring the recompilation of the SNMP agent. Refer to Chapter 7, “OS/400 SNMP Subagent” on page 91 for more information on SNMP subagent.

3.4 OS/400 SNMP General Overview

Figure 13 provides a general overview of the SNMP components and how they reference MIBs.

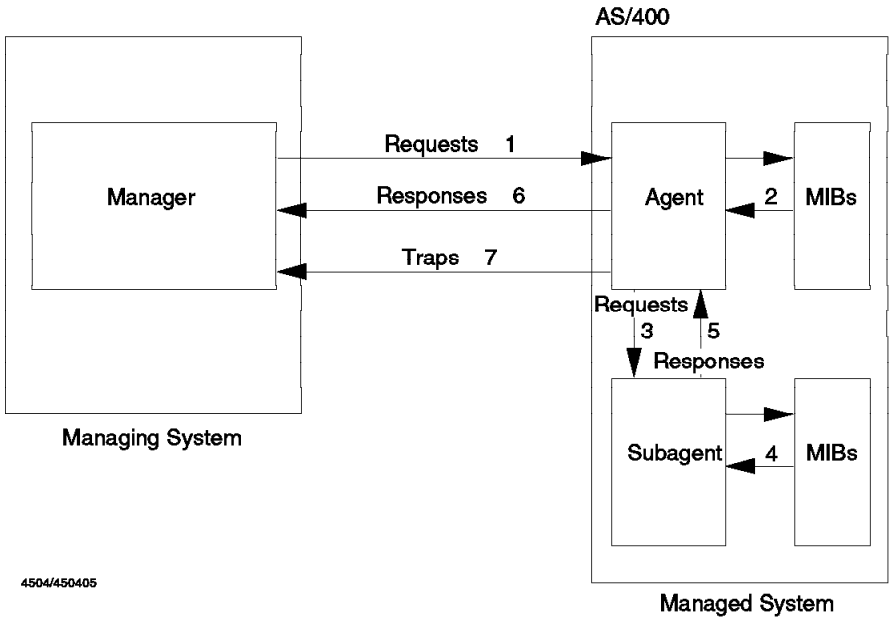


Figure 13. SNMP Overview

The type of protocol used by SNMP to pass data between the SNMP manager and the SNMP agent is a request/response protocol.

Step 1: The manager places a request for MIB object information.

Step 2: The agent gathers the information, requested by the manager, via the MIB. If the request is for a MIB-II variable there is no need to pass the request to the subagent and it continues with Step 6.

Step 3: If the request is for a variable contained in one of the registered subagents, then the request is sent to the subagent.

Step 4: The subagent gathers the information, requested by the agent, via the MIB.

Step 5: The subagent responds to the agent request with the information gathered in step 4.

Step 6: The agent responds to the manager request with the information gathered in step 2 or step 4.

Step 7: Traps are sent from the agent to the manager as unsolicited messages. A trap sent from the agent to the manager could be an authentication failure.

Chapter 4. Getting Started

This chapter is intended to help you configure SNMP on the AS/400. Once the configuration has been completed, we can then use an SNMP manager, such as NetView for OS/2, to verify the AS/400 SNMP agent.

The test environment used in the configuration examples is shown in Figure 14.

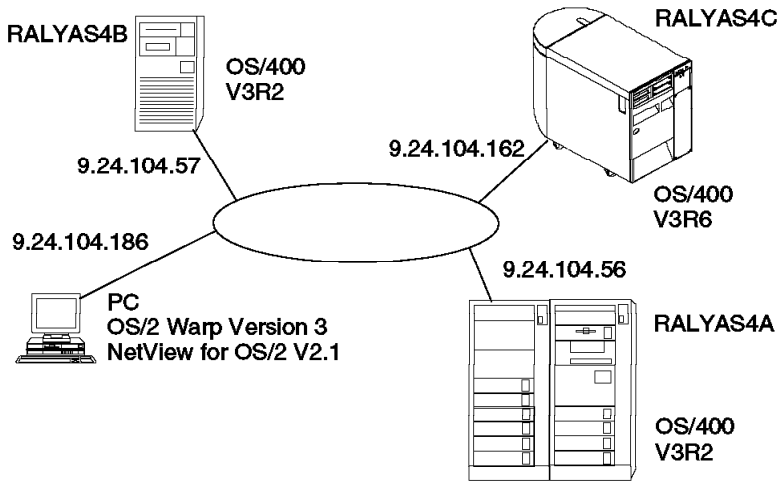


Figure 14. Our Lab Environment

4.1 TCP/IP Considerations

The SNMP protocol uses the functions provided by the lower layers of the TCP/IP protocol suite. This book does not cover the configuration of these lower layers and therefore assumes that a working TCP/IP connection is already in place between the manager and agent.

A quick way to verify the TCP/IP connection between the two systems is to use the TCP/IP `PING` command.

4.2 Verifying a TCP/IP Connection

PING is a TCP/IP application that sends data to a specific internet address and waits for a response. It then displays status information that indicates whether the connection was successful (response received) or unsuccessful (no response received).

To verify the connection between your AS/400 and the remote system (in this case a PC running NetView for OS/2 and TCP/IP) you will need to obtain the internet address of that remote system and follow the example below.

Go to an AS/400 command line and type the following:

PING

Press F4 and enter the internet address of the remote system as shown in Figure 15.

```
                          Verify TCP/IP Connection (PING)

  Type choices, press Enter.

  Remote system  . . . . . .   9.24.104.186

                                                                        Bottom
  F3=Exit   F4=Prompt   F5=Refresh   F10=Additional parameters   F12=Cancel
  F13=How to use this display        F24=More keys
```

Figure 15. Verifying the Connection to the Manager

Press Enter and wait until the verification completes.

On the command line enter the following command:

DSPJOBLOG

Press F10 to display the detailed messages.

A successful connection will produce a joblog like the one shown in Figure 16 on page 33.

```
                            Display All Messages
                                                              System:   RALYAS4A
Job . . :   WTRO5179E     User . . :   LEE           Number . . . :   021311

3 > PING RMTSYS('9.24.104.186')
    Verifying connection to host system 9.24.104.186.
    Connection verification 1 took .100 seconds.  1 successful connection
      verifications.
    Connection verification 2 took .019 seconds.  2 successful connection
      verifications.
    Connection verification 3 took .019 seconds.  3 successful connection
      verifications.
    Connection verification 4 took .019 seconds.  4 successful connection
      verifications.
    Connection verification 5 took .023 seconds.  5 successful connection
      verifications.
    Round-trip (in milliseconds) min/avg/max = 19/36/100
    Connection verification statistics: 5 of 5 successful (100 %).

Press Enter to continue.                                        More.......

F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```

Figure 16. A Joblog of a Successful Connection Verification

An unsuccessful connection will produce a joblog like the one shown in Figure 17.

```
                            Display All Messages
                                                              System:   RALYAS4A
Job . . :   WTRO5179E     User . . :   LEE           Number . . . :   021313




    Job 021313/LEE/WTRO5179E started on 07/05/95 at 10:21:28 in subsystem
      QINTER in QSYS. Job entered system on 07/05/95 at 10:21:28.
  > /*      */
3 > PING RMTSYS('9.24.104.222')
    Verifying connection to host system 9.24.104.222.
    No response from host within 10 seconds for connection verification 1
    No response from host within 10 seconds for connection verification 2
    No response from host within 10 seconds for connection verification 3
    No response from host within 10 seconds for connection verification 4
    No response from host within 10 seconds for connection verification 5
    Connection verification statistics: 0 of 5 successful (0 %).

Press Enter to continue.
                                                                    More....
F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```

Figure 17. A Joblog of an Unsuccessful Connection Verification

If the connection was verified successfully then you should proceed with the SNMP configuration.

If the connection was not successfully verified then you may have a hardware or, more likely, a configuration problem which your network administrator should fix before you proceed with the SNMP configuration.

4.3 Configuring the AS/400 SNMP Agent

In the following steps we configure the AS/400 SNMP agent.

To configure the AS/400 SNMP agent, enter the following command:

```
CFGTCPSNMP
```

You are presented with the following panel.

```
                              Configure TCP/IP SNMP
                                                          System:   RALYAS4A
Select one of the following:

     1. Change SNMP attributes
     2. Work with communities for SNMP









Selection or command
===> 1

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
```

Figure 18. Configure TCP/IP SNMP

4.3.1 Configuring SNMP Attributes

Take option 1 to change SNMP attributes.

```
                        Change SNMP Attributes (CHGSNMPA)

 Type choices, press Enter.

 System contact . . . . . .   'Lee Hargreaves. Raleigh ITSO. Tele# 0123 45
6 7890 (daytime). Pager# 098 765 4321 (outside working hours). Alternate contact
s - Francisco Infante & Maria Hermelo.'

 System location  . . . . . .   'International Technical Support Org. Ralei
gh Center  4912 Green Road  Raleigh, NC 27604'


 Send authentication traps  . . .   *YES          *SAME, *YES, *NO
 Automatic start  . . . . . .   *YES          *SAME, *YES, *NO
 Object access  . . . . . .   *NONE         *SAME, *READ, *WRITE, *NONE
 Log set requests . . . . . .   *NO           *SAME, *YES, *NO
 Log get requests . . . . . .   *NO           *SAME, *YES, *NO
 Log traps  . . . . . .   *YES          *SAME, *YES, *NO

                                                                      More....

 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

```
                        Change SNMP Attributes (CHGSNMPA)

 Type choices, press Enter.

 Trap managers:
   Manager internet address . . .   '9.24.104.186'
   Community name . . . . . .   'REDBOOK'



   Translate community name . . .   *YES          *YES, *NO
                + for more values









                                                                      Bottom

 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

Figure 19. Change SNMP Attributes

The parameters on this screen determine the facilities and behavior of the AS/400 SNMP agent.

System contact This field should contain the name and contact information of the person responsible for the operation of this AS/400. The data entered here can be retrieved by a managing system by viewing the *sysContact* MIB object of MIB-II. If you choose to enter *CNTINF in this field, the managing system will then retrieve the data entered in the service contact information. You can view this information by entering the following command:

```
WRKCNTINF
```

System location This field should contain the physical location of the AS/400. The data entered here can be retrieved by an SNMP manager by viewing the *sysLocation* MIB object of MIB-II.

Send authentication traps This field enables the AS/400 to report, without any request from a manager, any authentication failures to the manager. That is any incoming request from a manager not belonging to a community that is recognized by the AS/400 SNMP agent is reported.

> **Note**
>
> We suggest you use *YES as it provides a degree of security auditing.

Automatic start This field determines whether the SNMP agent is started when TCP/IP is started. If it is not, you must then run the following command:

```
STRTCPSVR SERVER(*SNMP)
```

Object access This field allows you to decide whether SNMP managers that are part of a recognized community have read only, read and write or no access rights (*READ, *WRITE or *NONE) to MIB objects on this system. The value entered here can be either used by, or overridden by, the same parameter in the community attributes.

> **Note**
>
> The default value in the community attributes is *SNMPATR which picks up the value you specify here. For security reasons it may be advisable to specify *NONE in the SNMP attributes and use the community attributes to grant specific object access rights.

Log set requests See Log trap requests.

Log get requests See Log trap requests.

Log traps The Log set requests, Log get requests and Log traps parameters specify whether these three activities are logged in a system journal, QUSRSYS/QSNMP. We look at this journal in more detail in Chapter 12, “Journal for SNMP Logging” on page 211.

Trap managers The Trap managers parameter specifies which SNMP managers will receive the traps generated by this system. You may need to ask your network administrator for the internet address and community name of the system to which traps are to be sent. Note that we have entered the internet address of the NetView for OS/2 system with a community name of redbook. You can specify multiple managers in this field.

Translate community name This field is set to *YES if the community name should be translated into ASCII characters before it is placed into the trap. Specify *YES if all the characters can be translated into ASCII. This would be appropriate if you were sending a trap to an ASCII based system.

After you have entered your desired values, press Enter to return to the Configure TCP/IP SNMP screen shown in Figure 20 on page 37.

4.3.2 Configuring SNMP Communities

A community describes a number of SNMP systems grouped by a community name. It comprises a list of SNMP managers and object access specifications. The community is used by an agent to determine which SNMP manager's requests should be honored by the AS/400 and what object access rights those managers have.

In this example we first verify the configuration of the community name *public* and then we configure a private community to which we grant higher object access capabilities.

The community name *public* is system supplied by default and allows any manager, in the *public* community, READ access rights to MIB objects on your system.

```
                          Configure TCP/IP SNMP
                                                          System:   RALYAS4A
Select one of the following:

     1. Change SNMP attributes
     2. Work with communities for SNMP









Selection or command
===> 2

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
```

Figure 20. Option 2 to Work with Communities

From the Configure TCP/IP SNMP screen, take option 2 to work with communities for SNMP.

```
                        Work with Communities for SNMP
                                                              System:   RALYAS4A
Type options, press Enter.
  1=Add   2=Change   4=Remove   5=Display

Option  Community Name

  5     public

                                                                          Bottom
F3=Exit   F5=Refresh   F6=Print list   F12=Cancel   F17=Top   F18=Bottom
```

Figure 21. Changing the Public Community

Take option 5 (Display) against the community *public*.

```
                          Display Community for SNMP
                                                              System:   RALYAS4A
Community name  . . . . . . . . . . . . . :   public



Translate community name  . . . . . . . . :   *YES
Object access . . . . . . . . . . . . . . :   *READ
Log set requests  . . . . . . . . . . . . :   *NO
Log get requests  . . . . . . . . . . . . :   *NO

Manager                    Manager                    Manager
Internet                   Internet                   Internet
Address                    Address                    Address
*ANY



                                                                          Bottom
Press Enter to continue.

F3=Exit   F12=Cancel
```

Figure 22. Display Community for SNMP

Using the object access parameter you can specify what rights SNMP managers using this community name have to the MIB objects on your system. We have left the community name *public* with the default *READ access by any (*ANY) SNMP manager.

Next we create our own private community.

```
                        Work with Communities for SNMP
                                                              System:   RALYAS4A
Type options, press Enter.
  1=Add   2=Change   4=Remove   5=Display

Option  Community Name
  1     REDBOOK
        public

                                                                          Bottom
F3=Exit   F5=Refresh   F6=Print list   F12=Cancel   F17=Top   F18=Bottom
Community changed.
```

Figure 23. Work with Communities for SNMP. Creating your own community.

Take option 1 and enter your community name.

```
                      Add Community for SNMP (ADDCOMSNMP)

Type choices, press Enter.

Community name . . . . . . . . . > 'REDBOOK'



Translate community name . . . .   *YES          *YES, *NO
Manager internet address . . . .   9.24.104.186
               + for more values
Object access  . . . . . . . . .   *write        *SNMPATR, *READ, *WRITE
Log set requests . . . . . . . .   *yes          *SNMPATR, *YES, *NO
Log get requests . . . . . . . .   *yes          *SNMPATR, *YES, *NO




                                                                          Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 24. Add Community for SNMP

The parameters on this screen determine which SNMP managers, when using the community name *redbook*, can gain access to this system's MIB objects and what access rights they have to them.

Community name The community name parameter is used by the agent to determine whether it should honor a request from a manager. In some ways it is like a password, used by the manager, to gain access to the agent.

Translate community name The translate community name parameter specifies whether or not to translate the community name into ASCII characters before it is compared to the community name imbedded in a request from a manager.

Manager internet address The manager internet address parameter identifies, by internet address, those managers that are part of this community. You may need to consult the network administrator of the SNMP manager to obtain this address. Note that we have entered the internet address of the NetView for OS/2 system.

Object access The object access parameter specifies, for all the managers listed in this community, whether they have read only, read and write or no access rights (*READ, WRITE or *NONE) to MIB objects on this system. Note that we have specified **write* access. This will allow the NetView for OS/2 system to perform *set* functions when this community name is specified in the request. Choosing a value of *SNMPATR will pick up the value specified in the SNMP attributes which you previously configured.

Note

By using several communities, with varying object access rights, it is possible to grant different managers higher or lower object access rights by assigning them to a different community.

The log set requests and log get requests parameters tell the AS/400 whether to log these requests from a manager in a system supplied journal QSNMP in QUSRSYS. We examine this journal more closely in Chapter 12, “Journal for SNMP Logging” on page 211.

Press Enter to complete the agent configuration.

4.3.3 Starting the AS/400 SNMP Agent

The AS/400 SNMP agent can be started either by default when TCP/IP is started or via a Start TCP/IP Server command. To start the SNMP agent by default when TCP/IP is started, automatic start must be set to *YES in the SNMP Attributes. To start the SNMP agent via a command, enter the following:

```
STRTCPSVR SERVER(*SNMP)
```

Chapter 5. NetView for OS/2

In this chapter we look at some of the options available within NetView for OS/2 and how to use them to retrieve information from an SNMP agent (in this case an AS/400).

5.1 Using NetView for OS/2 as an SNMP Manager

Three SNMP managers are available from IBM:

- NetView for OS/2
- NetView for AIX
- NetView for Windows

We chose NetView for OS/2 as our SNMP manager in the following examples. We don't, however, give a detailed description of NetView for OS/2. We only give a high level view to introduce you to the functions provided by a typical SNMP manager.

5.1.1 The NetView for OS/2 Desktop

We make the assumption that NetView for OS/2 is already installed and configured to use TCP/IP as its management protocol.

From the OS/2 Desktop, open the IBM NetView for OS/2 folder by double-clicking on the icon.

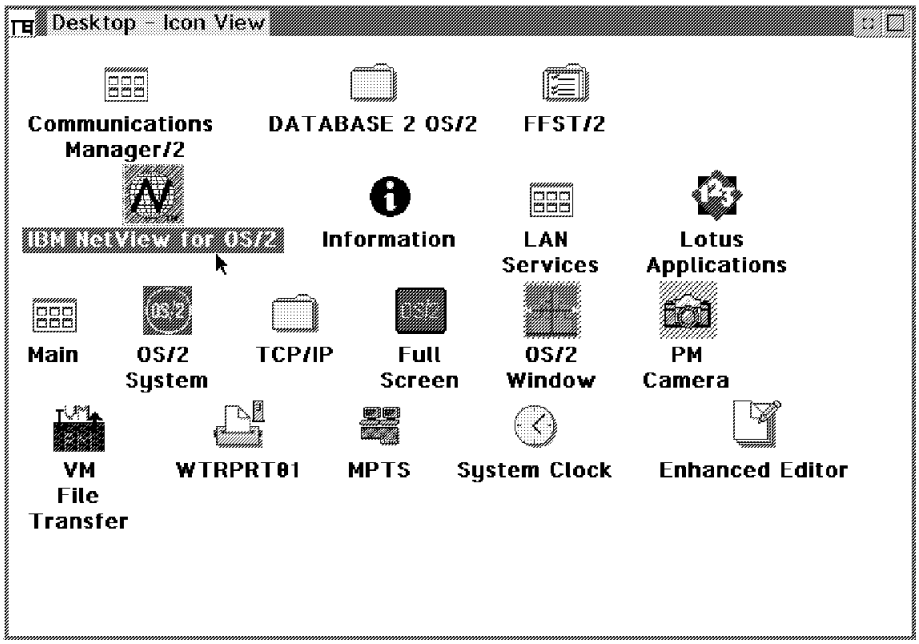


Figure 25. OS/2 Desktop

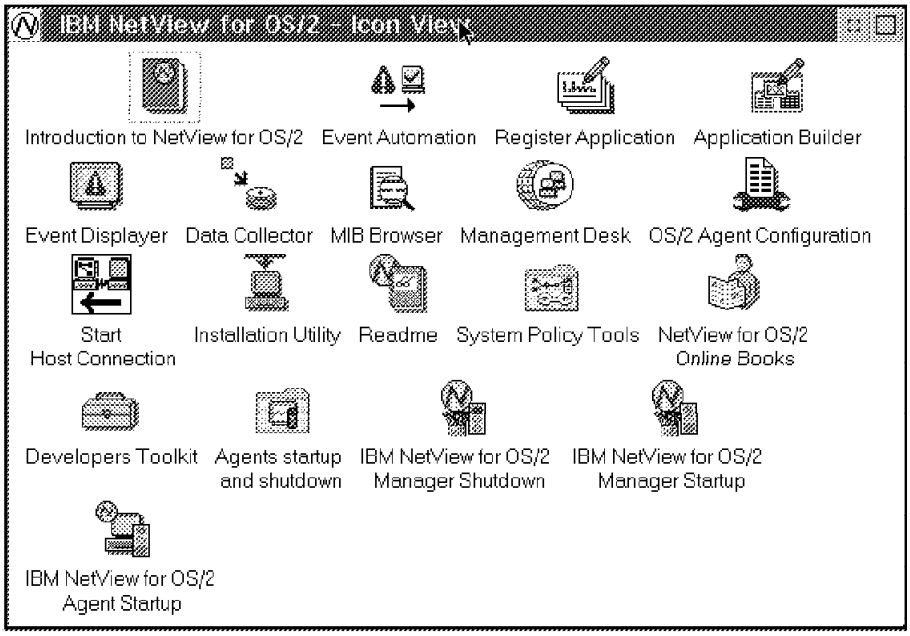


Figure 26. NetView for OS/2 Folder

In this book we will only explore the functions behind a few of these icons. However, below is a summary of the functions that lie behind all these icons.

Developers Toolkit The developer's toolkit is a collection of application program interfaces (APIs) that allow the development of management programs. It includes sample programs and files that can be used by a developer to create programs for the management desk.

Introduction to NetView for OS/2 The Introduction to NetView for OS/2 folder provides brief summary of the product and its functions.

NetView for OS/2 Online Books The online books folder, as the title suggests, provides online documentation about NetView for OS/2. The documentation includes NetView for OS/2 Installation and Admin Guide, NetView for OS/2 User Guide, and NetView for OS/2 Agent Guide.

Readme The Readme file contains tips about such topics as installation and troubleshooting.

Installation Utility The installation utility is used to: add additional components, update existing components, reload files or icons that were lost or remove NetView for OS/2.

Start Host Connection The Host Connection application allows host NetView (where host NetView refers to NetView for MVS, VM and VSE) to receive traps from systems managed by NetView for OS/2. Communications Manager/2 acts as a bridge between NetView for OS/2 and host NetView.

OS/2 Agent Configuration The OS/2 agent configuration allows you to configure the system as an SNMP agent. MIB variables such as sysLocation and sysDesc can be entered here for subsequent retrieval by an SNMP manager.

Management Desk The management desk can be used by a system administrator to visually monitor and manage SNMP agents. We will cover some of the functions of the management desk in more detail later.

MIB Browser The MIB browser can be used to retrieve the values of MIB objects on an SNMP agent quickly and easily. It can display the retrieved value as text or in graphical form. We will be using the MIB browser extensively later in this chapter.

Data Collector The data collector allows you to create your own collection configurations. It can be used to collect specific MIB variables from a variety of systems at predetermined intervals.

Event Displayer The event displayer is used to monitor SNMP traps that are stored in the trap log file. It can display all traps or a subset of traps based on user-defined trap filtering.

Application Builder The application builder allows you to create and customize a MIB application which retrieves specific MIB variables when the application is run. The application you create is stored in the MIB Applications folder. Using this feature, it is possible to build applications that can be run when there is a system problem. For example, you could build an application that constantly retrieved the system CPU usage value. Then, at times of poor performance, run the application and display the output graphically. This may tell you if the performance problem is CPU related.

Register Application Register application is used to add the MIB applications you have built to the pull-down menu of system objects on the management desk.

Event Automation Event automation allows you to invoke an action such as sending a message to the screen when a specific type of trap is received.

IBM NetView for OS/2 Manager Startup This icon is put into the Startup folder at installation time and is used to start the NetView for OS/2 Process Manager. The process manager handles the requests to start specific management programs. NetView for OS/2 can also be started by the command SVSTART. We will be using an example of this command later in the chapter.

IBM NetView for OS/2 Manager Shutdown This icon allows you to stop the IBM NetView for OS/2 Manager processes. The processes can also be stopped via the SVSTOP OS/2 command. We will be using an example of this command later in the chapter.

IBM NetView for OS/2 Agent Startup This icon is also put into the Startup folder at installation time. If your system is configured for agent support only, then only this icon will be present.

Agents startup and shutdown This icon provides the possibility to start or to stop agents processes manually. These processes are: Communication Manager/2 agent, LAN requester agent, SNMP daemon and system information agent.

System Policy Tools The system policy tools made available via this icon are: System policy configuration, Event log viewer and Transfer database viewer.

5.1.2 The Discovery Process

NetView for OS/2 discovery is the process by which NetView for OS/2 locates systems in the network. To do this, it monitors the line for IP traffic. Once a system has been discovered, it is polled on a regular basis to check whether the connection still exists.

Note

We found that to be certain of a system being discovered, it was best to PING it several times from an OS/2 window. It is possible, however, using Seed files and Mask files, to customize the discovery process. We created Seed and Mask files to ensure NetView for OS/2 attempted to discover three systems in our network and no more. Refer to Appendix E, "Customizing the NetView for OS/2 Discovery Process" on page 293 for an example of how to use Seed files and Mask files.

Before you can retrieve information from an agent, NetView for OS/2 must first have ″discovered″ it. Discovered systems are stored in the All Systems folder which we will come to later.

The discovery process starts by default when NetView for OS/2 is started. To see which processes are started, go to an OS/2 window and enter the following command:

```
SVSTATUS
```

If the NetView for OS/2 startup icons are not in the startup folder, then you may receive the following message:

```
NetView for OS/2 Process Manager not running
```

If they were in the startup folder, you will be presented with a screen showing a list of processes and their current status.

Note

We suggest you consider dragging the two startup icons from the startup folder and dropping them in the NetView for OS/2 Icon View main folder. This allows you to control the startup procedure manually.

In this book we will be concentrating on systems using SNMP over TCP/IP and, therefore, only need to discover systems using TCP/IP; although, it is possible for NetView for OS/2 to discover systems using IPX and NetBIOS protocols also.

The three processes that NetView for OS/2 uses for discovery are as follows:

tcpipdiscovery This process will discover systems in the network that are using the TCP/IP protocol. This is the process that we shall be using.

netbiosdiscovery This process will discover systems in the network that are using the NetBIOS protocol.

netwdiscovery This process will discover systems in the network that are using the IPX or NetWare protocol.

To start the discovery process, you will first need to start the Process Manager if you have removed the icons from the startup folder. Go to the IBM NetView for OS/2 icon view folder and double-click on the **IBM NetView for OS/2 Startup** icon.

The SVSTATUS command can be used to show the status of NetView for OS/2.

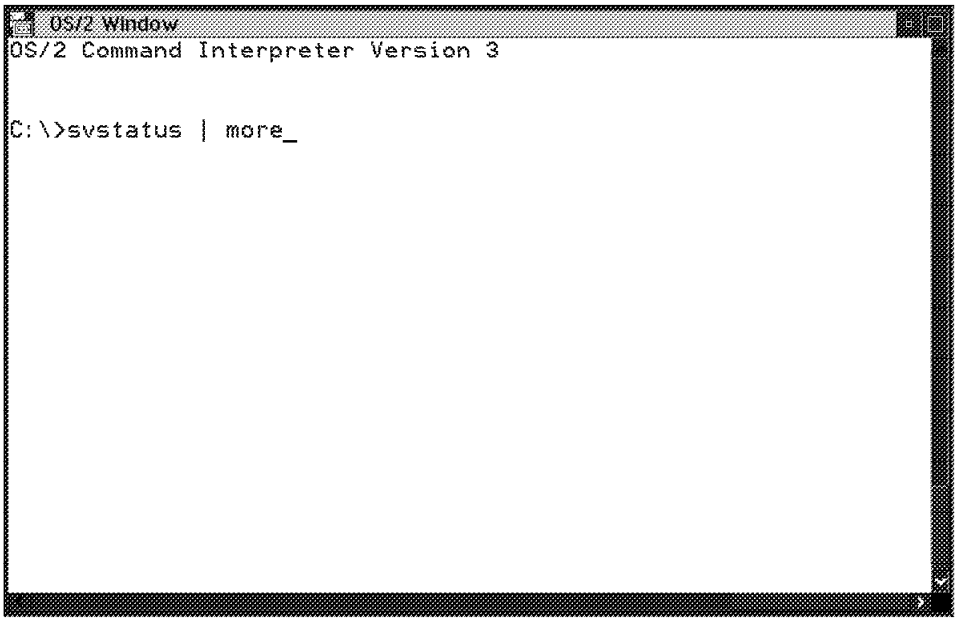


Figure 27. SVSTATUS Command using the "More" Extension

The more extension allows you to scroll through the process status screens. See Figure 28 for an example screen.

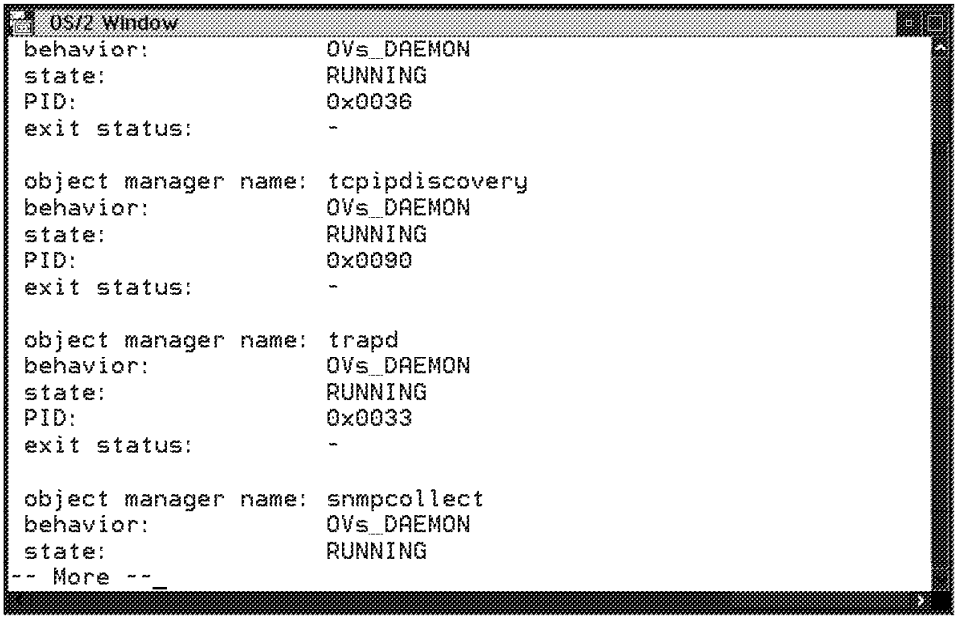


Figure 28. SVSTATUS Command Showing TCPIPDISCOVERY Process Running

Any of the processes can be stopped by entering the following command:

SVSTOP <process name>

See Figure 29 on page 46 for an example.

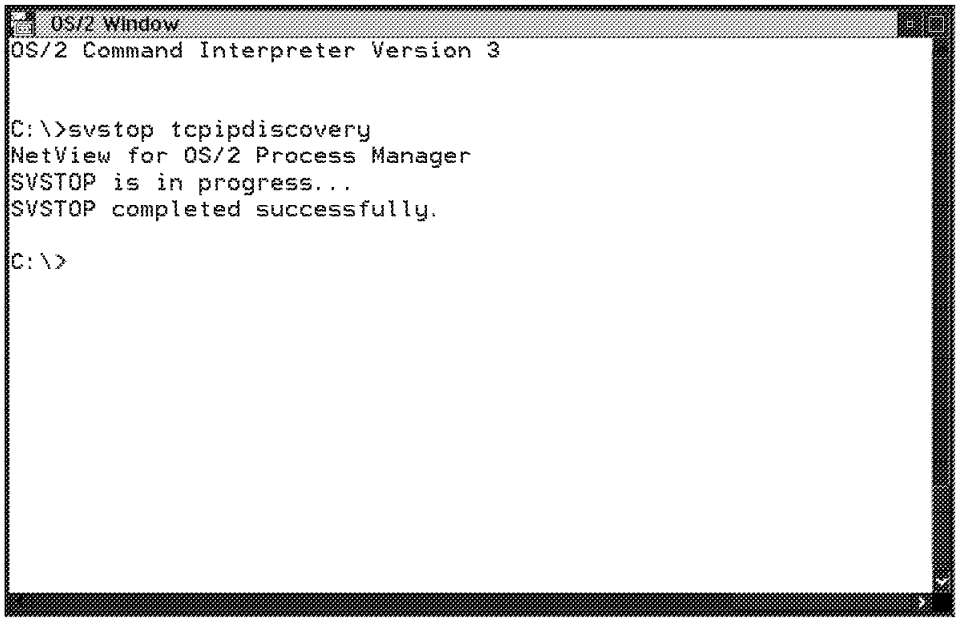


Figure 29. SVSTOP Command for Process TCPIPDISCOVERY

Because there may be, potentially, hundreds of TCP/IP systems that could be discovered by NetView for OS/2 and take up disk space on the PC, it may be desirable to limit the discovery of systems to ones that are relevant to your management domain. Seed files and Masks can be used for this purpose. Alternatively, you could simply stop the discovery process once all the systems you want to manage have been discovered (see Figure 29).

5.1.3 Using NetView for OS/2 to Retrieve Data

Now that we know something about the discovery process and the functions provided by NetView for OS/2, let's use some of them to access information on the AS/400 using the management functions of NetView for OS/2 and the agent functions of the AS/400.

Double-click on the **Management Desk** from the NetView for OS/2 icon view folder.

You should now see the NetView for OS/2 Product Information screen as shown in Figure 30 on page 47. Click on the **OK** button.

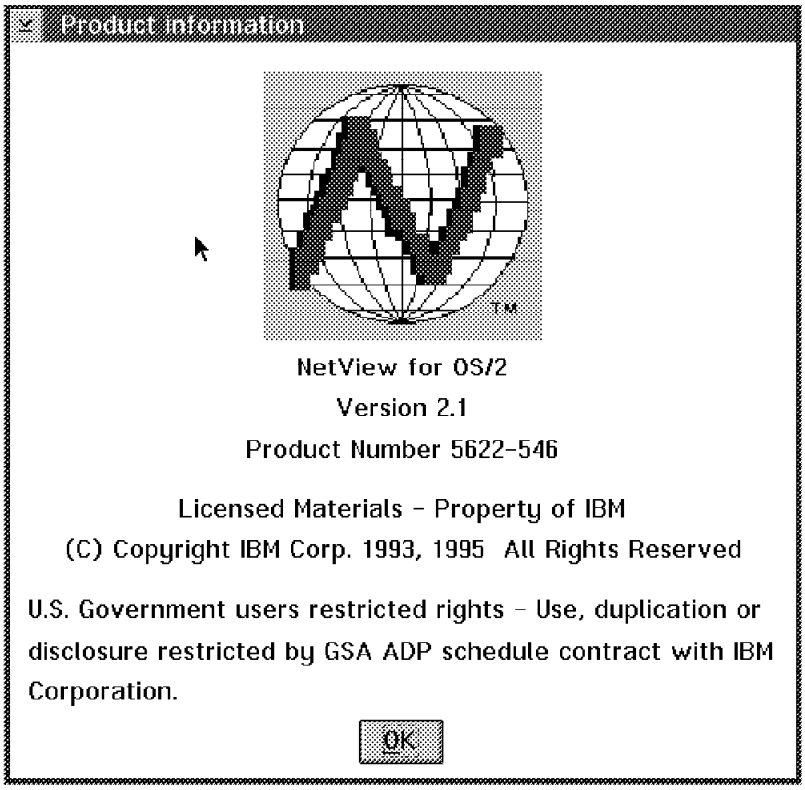


Figure 30. NetView for OS/2 Product Information

Wait while the Management Desk application loads.

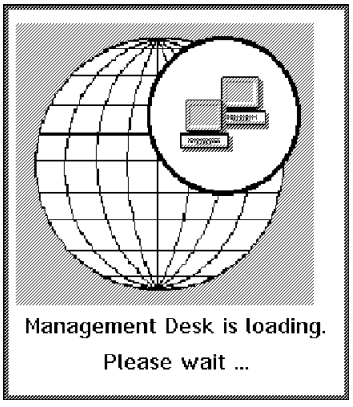


Figure 31. Management Desk Application Loading.....Wait

You should now see the Management Desk icon view as shown in Figure 32 on page 48.

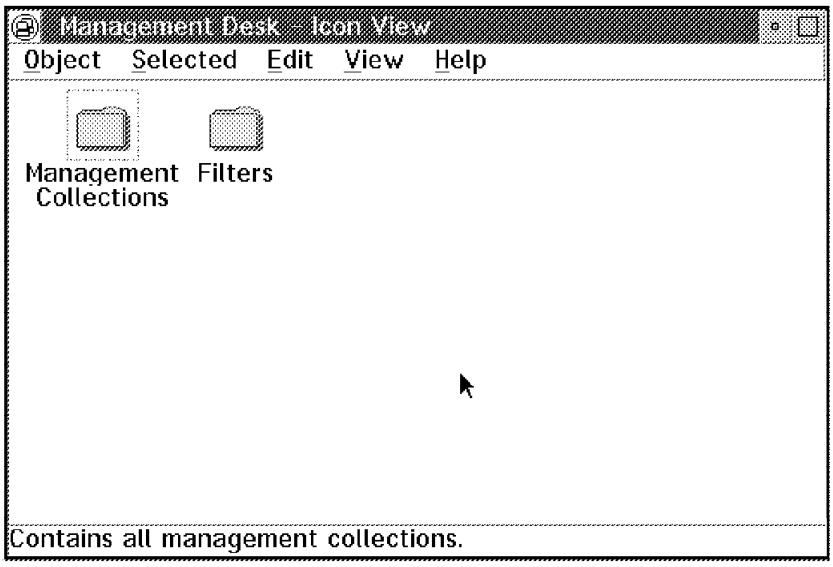


Figure 32. Management Desk Icon View

Double-click on the **Management Collections** icon. You should now see the Management Collections folder as shown in Figure 33.

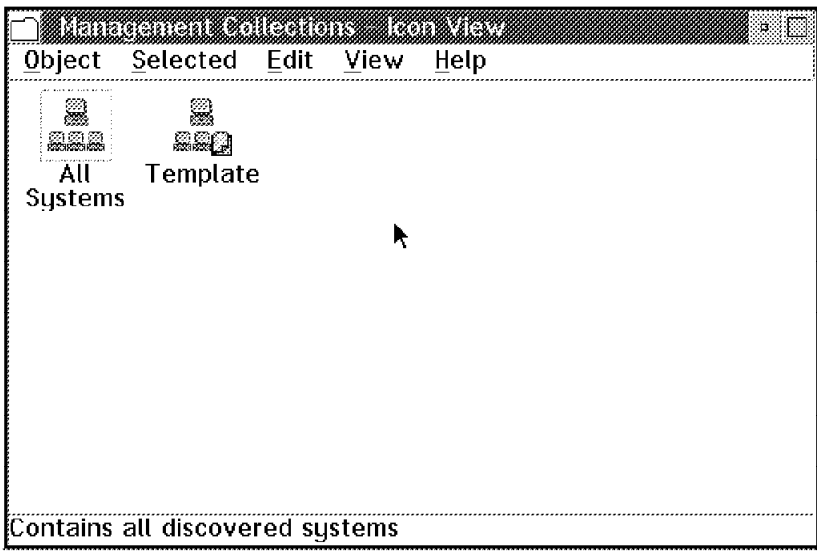


Figure 33. Management Collections icon view

At the Management Collections icon view double-click on the **All Systems** icon. You should now see the All System folder as shown in Figure 34 on page 49.

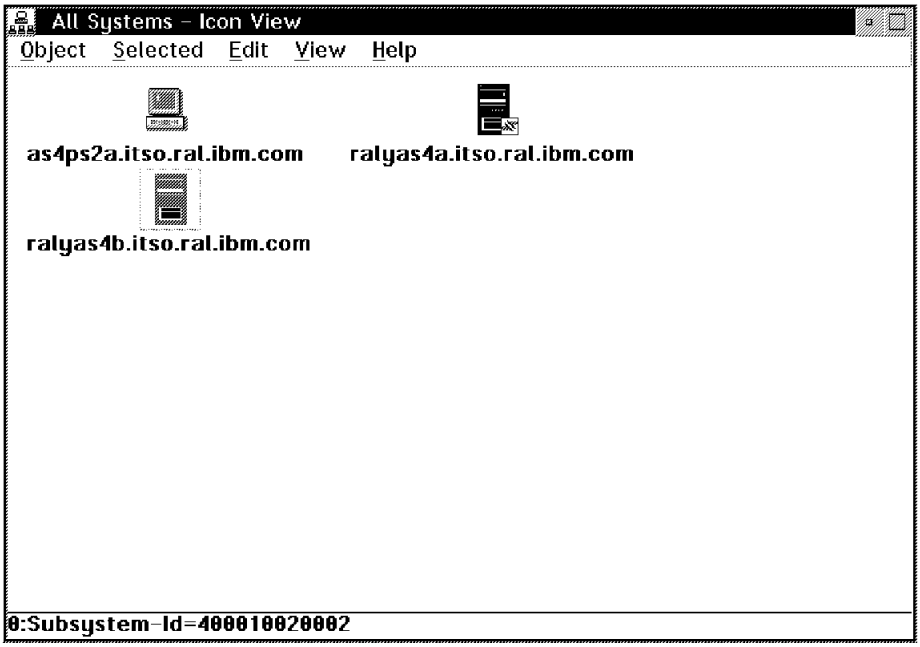


Figure 34. Three Systems Discovered

Once NetView for OS/2 has discovered the agent system, you can use the MIB Browser to retrieve information from it. In Figure 34 you can see that three systems were discovered: RALYAS4A, RALYAS4B and the PC running NetView for OS/2 (AS4PS2A).

In the following examples we used RALYAS4B because RALYAS4A was being IPLd and was therefore offline (indicated by the small symbol at the bottom right of the AS/400 icon).

Note

The AS/400 icons shown in Figure 34 were hand drawn using the icon editor. Because the icon becomes associated with the system name, it is possible to create an individual icon for each individual system. This makes it much easier to identify the systems on the discovery screen. To use the icon editor, click on the icon with the right mouse button and open settings. If you then click on the General page of the settings view you will see the option to edit the icon.

Figure 35 on page 50 and Figure 36 on page 50 show the AS/400 icon for RALYAS4A when it is not communicating with the manager and then when communications have been resumed, that is when the IPL has completed. Once the discovery process has discovered a system, the manager keeps polling it at regular intervals to verify that it is still communicating.

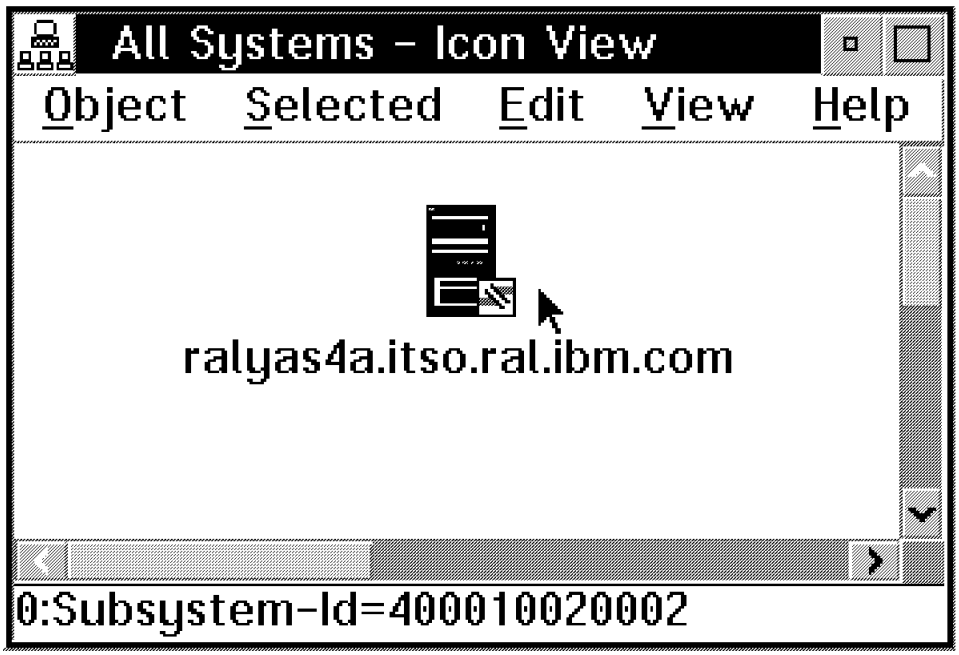


Figure 35. Communications Lost with RALYAS4A

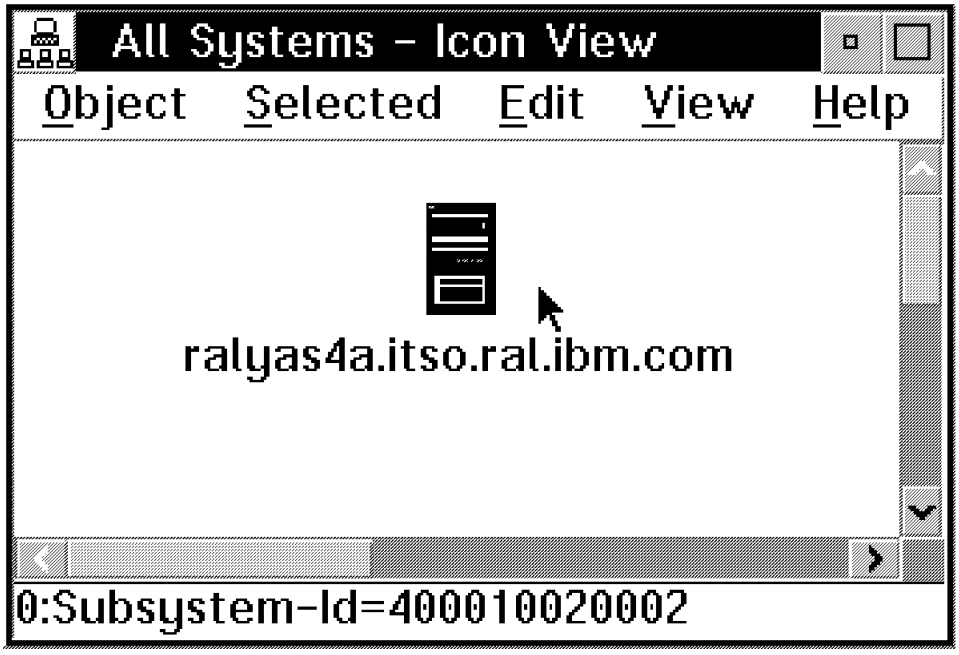


Figure 36. Communications Resumed with RALYAS4A

5.1.4 Using the MIB Browser

A MIB Browser is used to request MIB data from a managed system.

Go to the NetView for OS/2 folder and double-click on the **MIB Browser** icon. You should be presented with the MIB Browser control panel as shown in Figure 37 on page 51.

Alternatively you can access the MIB Browser for a specific system, from the All Systems folder shown in Figure 34 on page 49. Select the system by clicking once on its icon, using the right mouse button, then click on **Application action**. This should present you with a list of applications that can be started. Select **MIB Browser**.

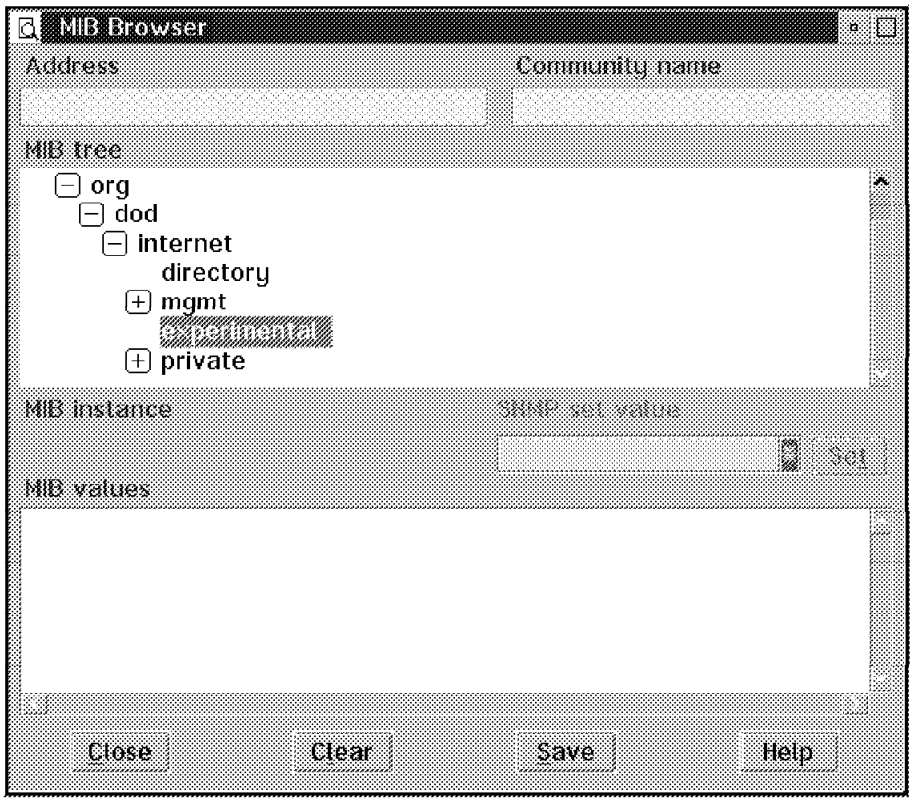


Figure 37. MIB Browser Control Panel

The MIB browser allows us to explore the MIB tree (see Figure 9 on page 17) by selecting a MIB object and using the *Plus* and *Minus* symbols. In the example that follows we will retrieve the value *sysContact* from system RALYAS4A.

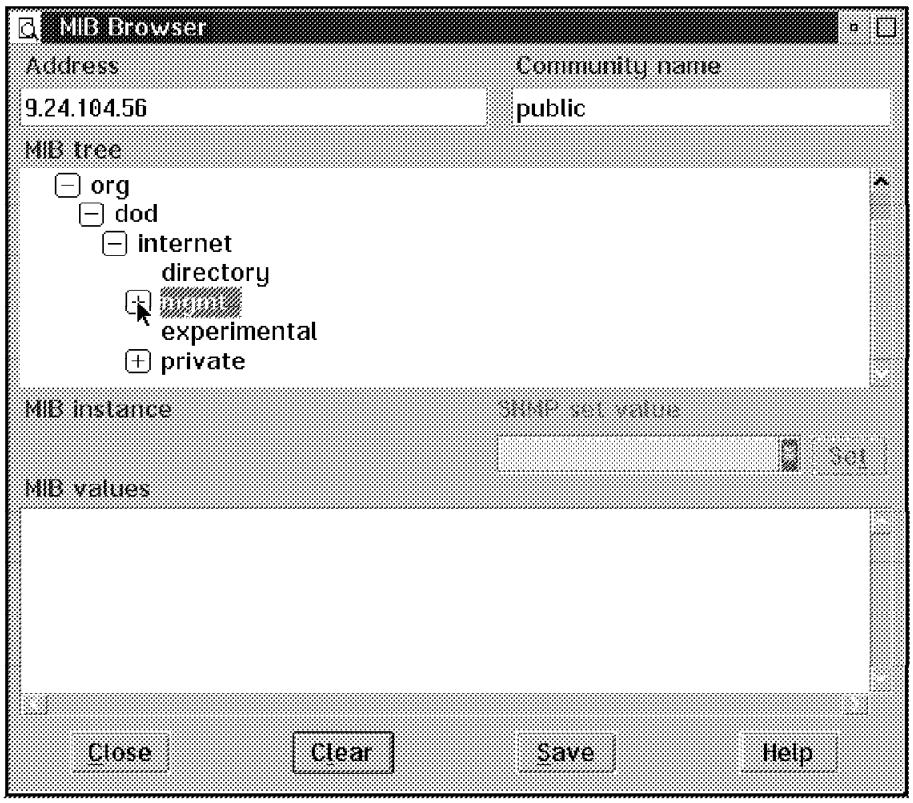


Figure 38. Entering the Address and Community of the Agent

Enter the IP address of the agent. In this case the address of RALYAS4A is 9.24.104.56.

Enter the community name that the AS/400 agent is a member of. In this case we have *READ authority to the *public* community on the AS/400. See Figure 22 on page 38.

Click on the **mgmt** object to expand the view of the mgmt subtree. See Figure 39 on page 53.

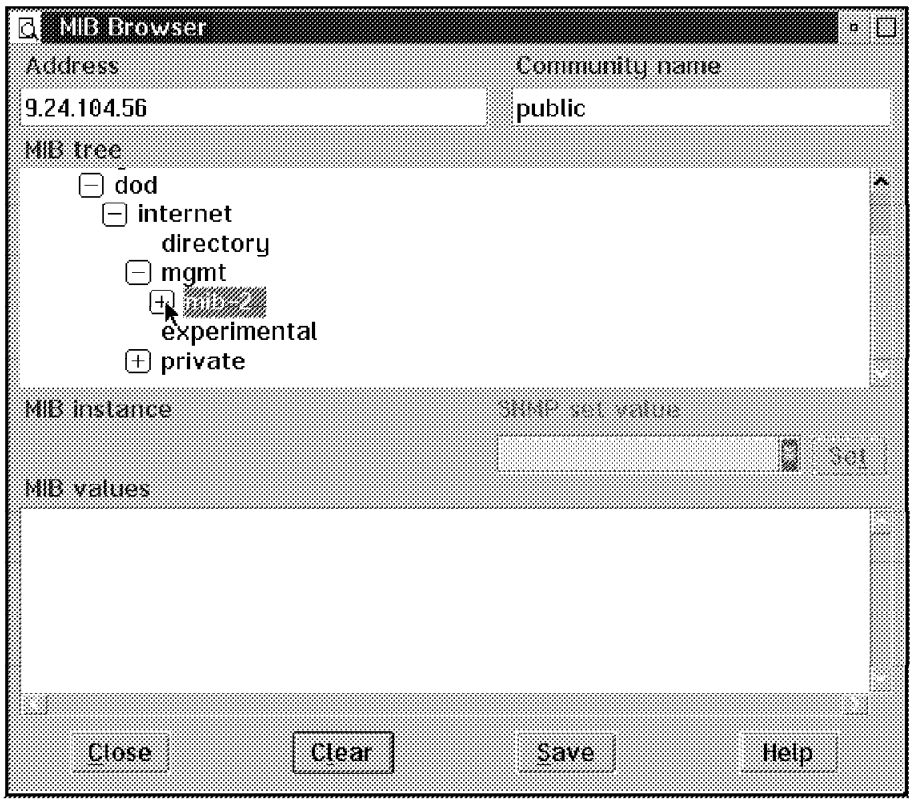


Figure 39. The MIB-2 Object

Now click on the **mib-2** object to expand the view of the mib-2 subtree. See Figure 40 on page 54.

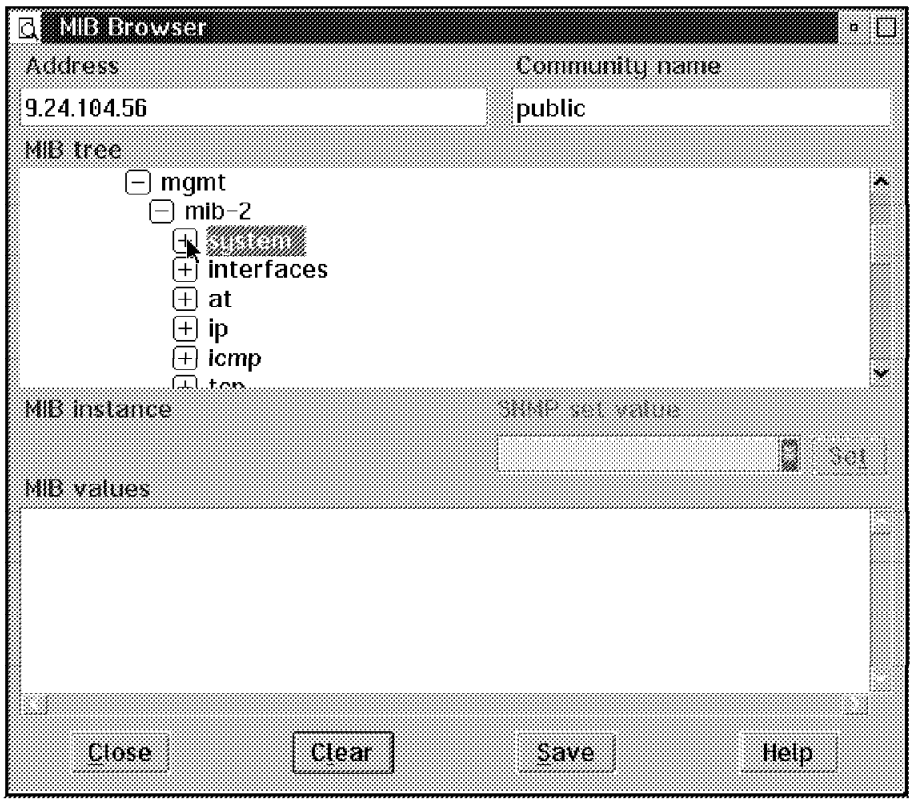


Figure 40. MIB-2 Groups

Click on the **system** object. Now you will see the "leaf" objects within the system group such as *sysDesc* and *sysContact*. See Figure 41 on page 55.

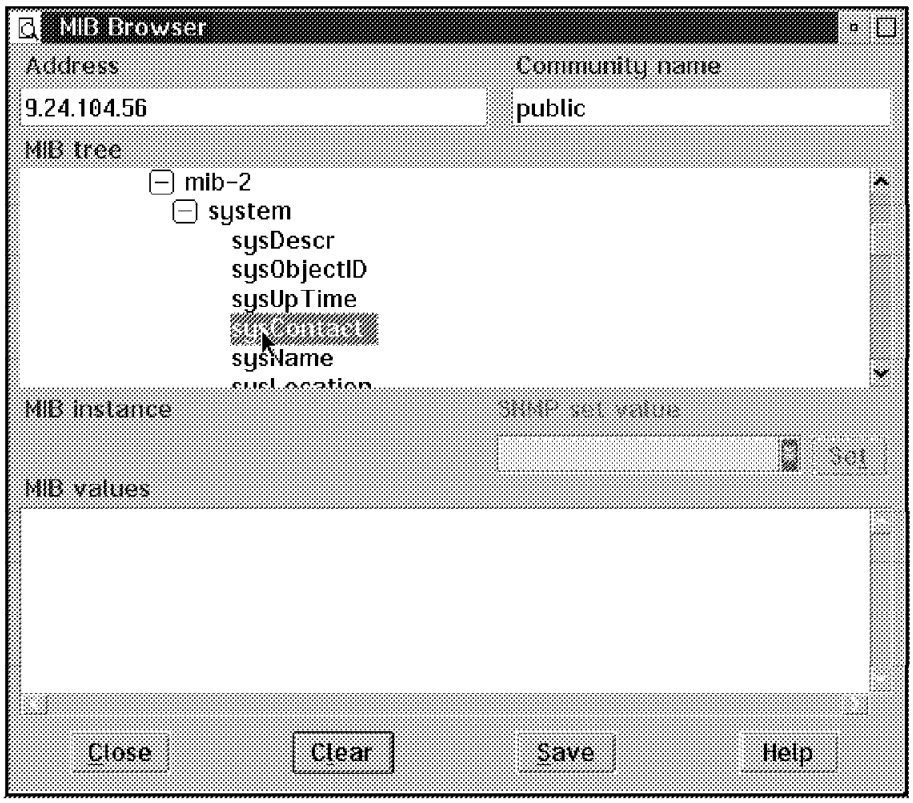


Figure 41. Using the Describe Feature

In this example we have selected the *sysContact* MIB object by clicking on it once. Click once on the right mouse button; you should be presented with a submenu. Before retrieving the MIB information, select **Describe** from the submenu. This will present a panel with a description of the *sysContact* variable, as well as the route you took down the MIB tree to get to this variable. See Figure 42. Alternatively You can double click over the system contact.

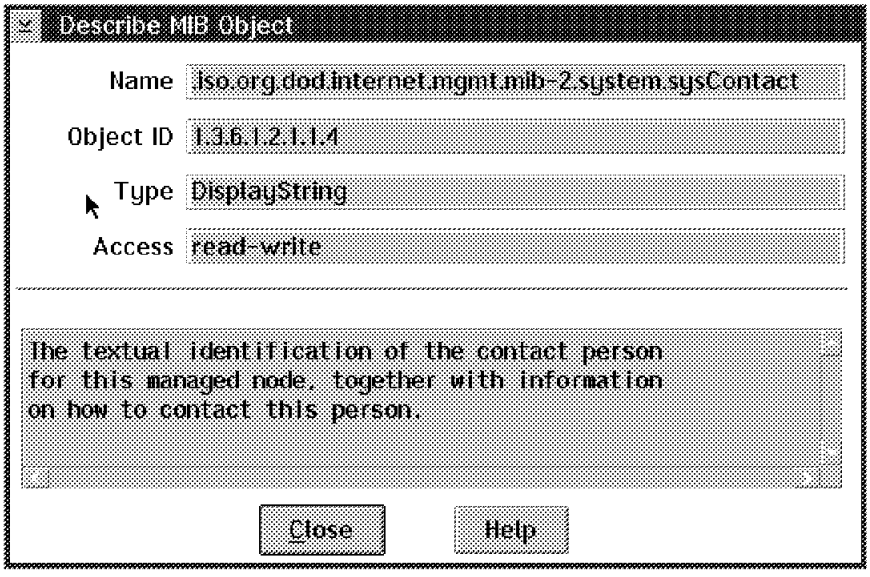


Figure 42. A Description of the Variable

Click on the **Close** button.

Return to the MIB browser panel; now select **Start query** from the submenu. This will send a request for *sysContact* information to the system whose IP address you entered earlier. If all has been configured correctly, you should receive the value stored in the system contact parameter in the SNMP attributes of the AS/400. See Figure 43 for the results and for the submenu displayed.

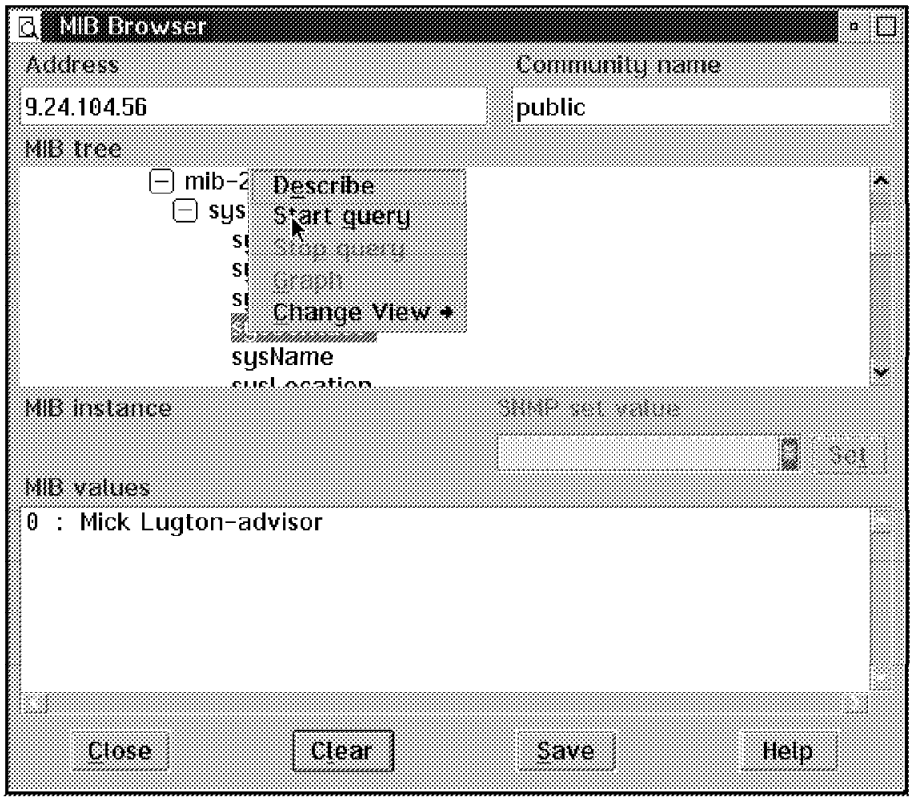


Figure 43. sysContact Variable Retrieved

In Figure 43 you can see the sysContact value returned. In Figure 44 on page 57 you can see the same value as configured in the AS/400's SNMP attributes.

```
                        Change SNMP Attributes (CHGSNMPA)

Type choices, press Enter.

System contact . . . . . .   'Mick Lugton-advisor'



System location  . . . . . .   'Raleigh ITSO - North Carolina'



Send authentication traps  . . .   *YES          *SAME, *YES, *NO
Automatic start  . . . . . .   *YES          *SAME, *YES, *NO
Object access  . . . . . .   *READ         *SAME, *READ, *WRITE,
Log set requests . . . . . .   *YES          *SAME, *YES, *NO
Log get requests . . . . . .   *YES          *SAME, *YES, *NO
Log traps  . . . . . .   *YES          *SAME, *YES, *NO
```

Figure 44. AS/400 SNMP Attributes

5.1.5 The NetView for OS/2 Grapher Function

Now you have retrieved a static value, the next example will show you how to retrieve a dynamic value then display it graphically. For this you will need to go into a different MIB.

Click on the **Up tree** button until you come to the panel showing the four MIB objects: directory, mgmt, experimental and private as shown in Figure 37 on page 51.

From here you will need to click on a sequence of MIB objects to take you down another branch of the MIB tree.

Click on the following from the MIB Browser in this sequence:

1. **private**
2. **enterprise**
3. **ibm**
4. **ibmProd**
5. **netView6000SubAgent**
6. **nv6saComputerSystem**

Now you should see the variable *nv6saComputerSystemLoad*.

Click on the **Start query** button to receive the value at that instant. The value returned should be divided by 100 to give a true percentage. For example, a value of 0 : 3850 would equate to 38.5% CPU.

Now, click on the **Graph** button and you should see a screen like the one in Figure 45 on page 58.

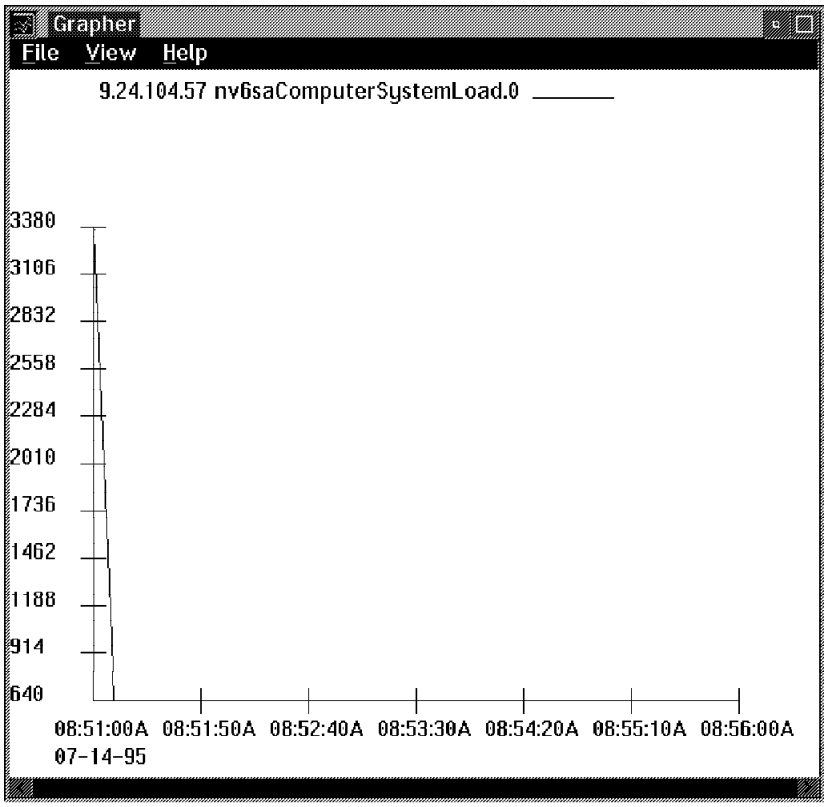


Figure 45. Graphical Representation of System Load

By using the **View** pull-down on this screen it is possible to change various characteristics of the graph function such as the time interval between samples and the time period shown by the x-axis.

In the example shown in Figure 46 on page 59, the graph function has been sampling the system load of RALYAS4B for about 5 minutes.

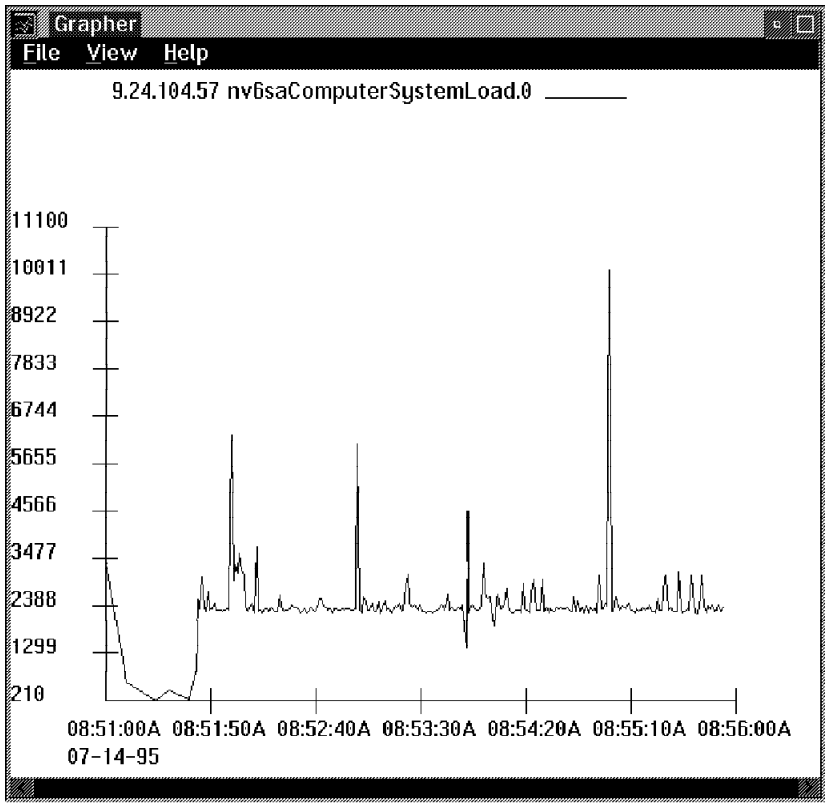


Figure 46. RALYAS4B CPU Load over a Five Minute Period

The time interval between samples in Figure 46 was set to one second and therefore shows even brief fluctuations in CPU load. It can be seen that the average CPU load over this five minute period is around 25 percent.

In the next example, shown in Figure 47 on page 60, the sample period starts at 8:09:42pm on the 13th of July and runs throughout the night and following morning.

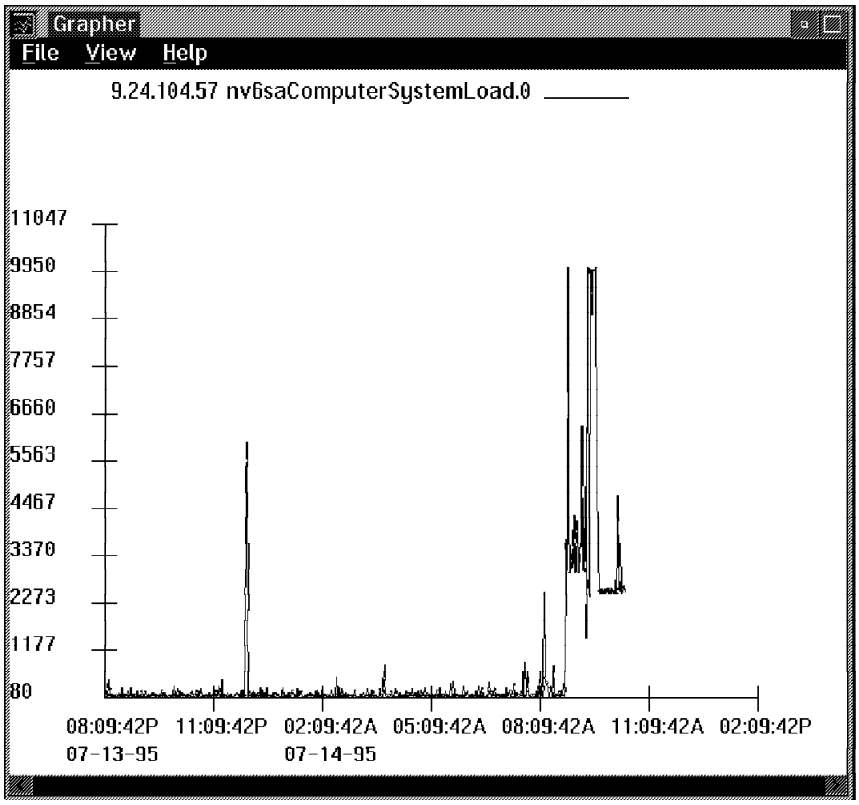


Figure 47. Processor Load for a Longer Period

It is clear that the processor load is very low for most of the night with the exception of a spike shortly after 11:09:42pm. Because the time interval between samples was set to one minute, any deviation in processor load may indicate a significant period of activity. Using the Time interval setting from the View pulldown, it is possible to examine instances like this more closely. In Figure 48 on page 61, we set the display width to ten minutes and then looked at the time period of the spike.

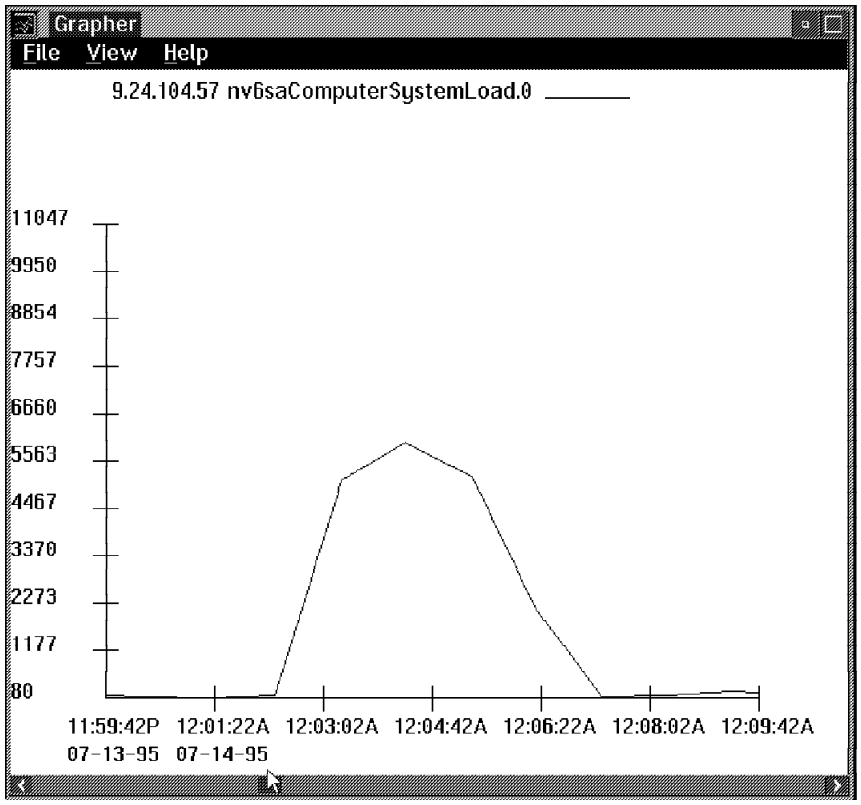


Figure 48. Using Some of the Grapher Options to Zoom-in

Now it is apparent that the processor load was running at around fifty percent between 12:03:02am and 12:05:00am. After checking the system history log, QHST, on the AS/400 we discovered that the system cleanup task was running during this time. Figure 47 on page 60 shows that the processor load remained relatively low until around 08:00:00am when users started signing on to the AS/400.

5.1.6 Using NetView for OS/2 to Set Information

As discussed in previous chapters, it is possible to set a value in a MIB variable as well as retrieve the data it contains. The next example explains how to set the variables in the MIB-2 group system.

Using the procedures shown in Figure 38 on page 52 through Figure 43 on page 56 retrieve the *sysContact* variable again. Click on the retrieved MIB value. See Figure 49 on page 62.

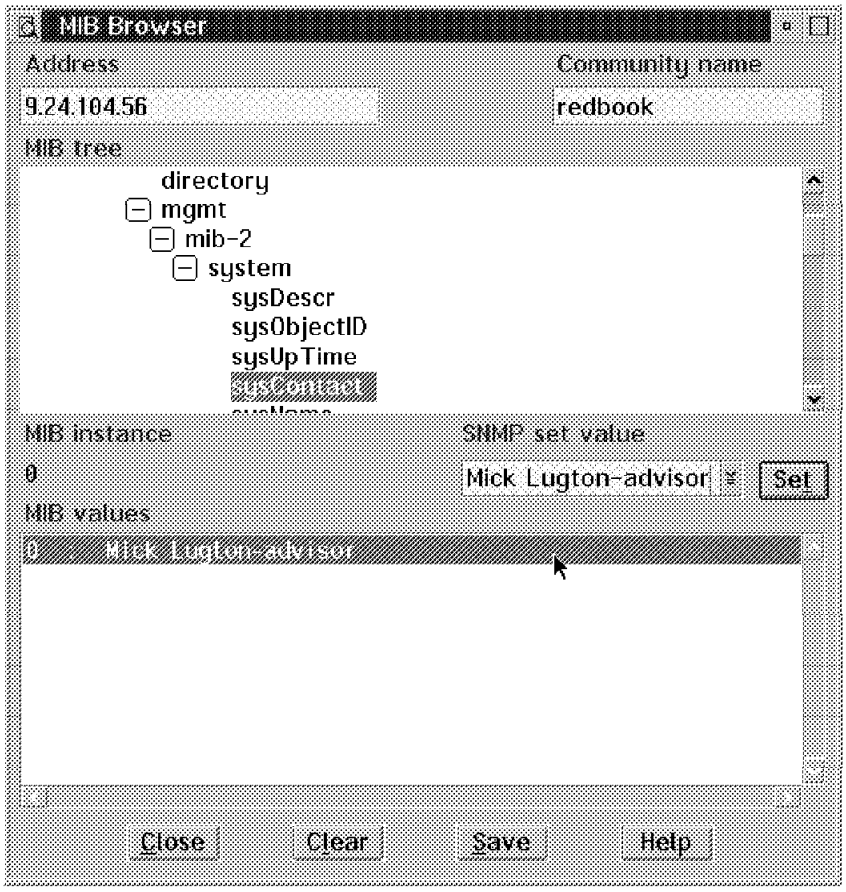


Figure 49. Selecting the MIB Variable to Set

Now the Set button should have become available. Type the new value in the SNMP set value field and click on the **Set** button. See Figure 50 on page 63.

Note

Notice that we are using the community name *redbook*. This community allows us *WRITE authority. If you are not using a community that allows you *WRITE authority then you will get a failure. See Figure 24 on page 39 for RALYAS4A *redbook* community configuration.

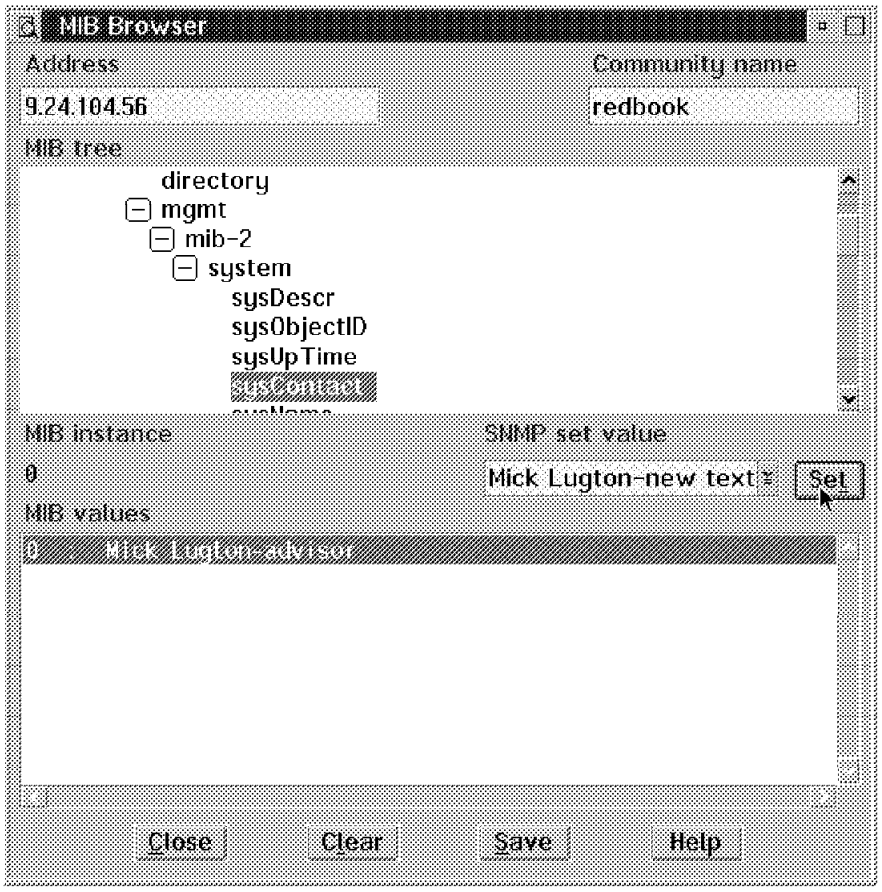


Figure 50. SETting the New Value

See Figure 44 on page 57 for the value of the *sysContact* variable before it is changed.

You should receive the `Set value successful message` as shown in Figure 51.

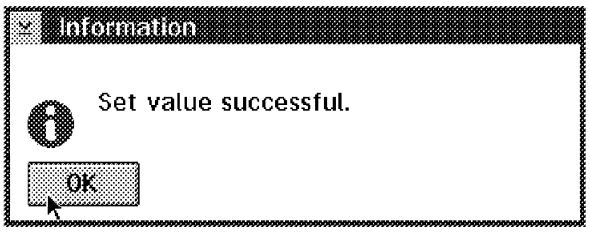


Figure 51. SET Completed Successfully

Now click on the **Start query** button once more to display the new MIB value. See Figure 52 on page 64.

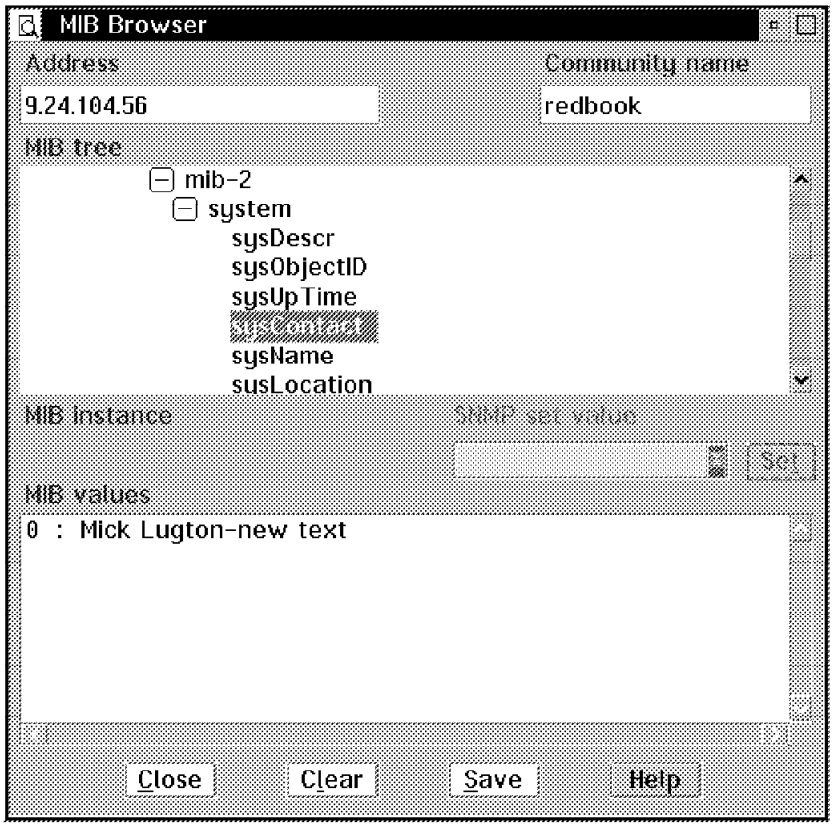


Figure 52. Retrieving the New Value

Figure 53 shows the resultant AS/400 SNMP attributes screen.

```
                    Change SNMP Attributes (CHGSNMPA)

Type choices, press Enter.

System contact . . . . . .   'Mick Lugton - new text'


System location  . . . . . .   'Raleigh ITSO - North Carolina'


Send authentication traps  . . .   *YES          *SAME, *YES, *NO
Automatic start  . . . . . .   *YES          *SAME, *YES, *NO
Object access  . . . . . .   *READ         *SAME, *READ, *WRITE,
Log set requests . . . . . .   *YES          *SAME, *YES, *NO
Log get requests . . . . . .   *YES          *SAME, *YES, *NO
Log traps  . . . . . .   *YES          *SAME, *YES, *NO
```

Figure 53. AS/400 SNMP Attributes Showing Changed sysContact Variable

5.1.7 Managing Multiple AS/400s

You have now seen how to view and set information on the AS/400, we will now look at how to use NetView for OS/2 to monitor multiple AS/400s and trigger an event when a certain condition arises.

Go to the IBM NetView for OS/2 icon view folder and double-click on the **Data Collector** icon. You will see the Data Collector screen as shown in Figure 54.

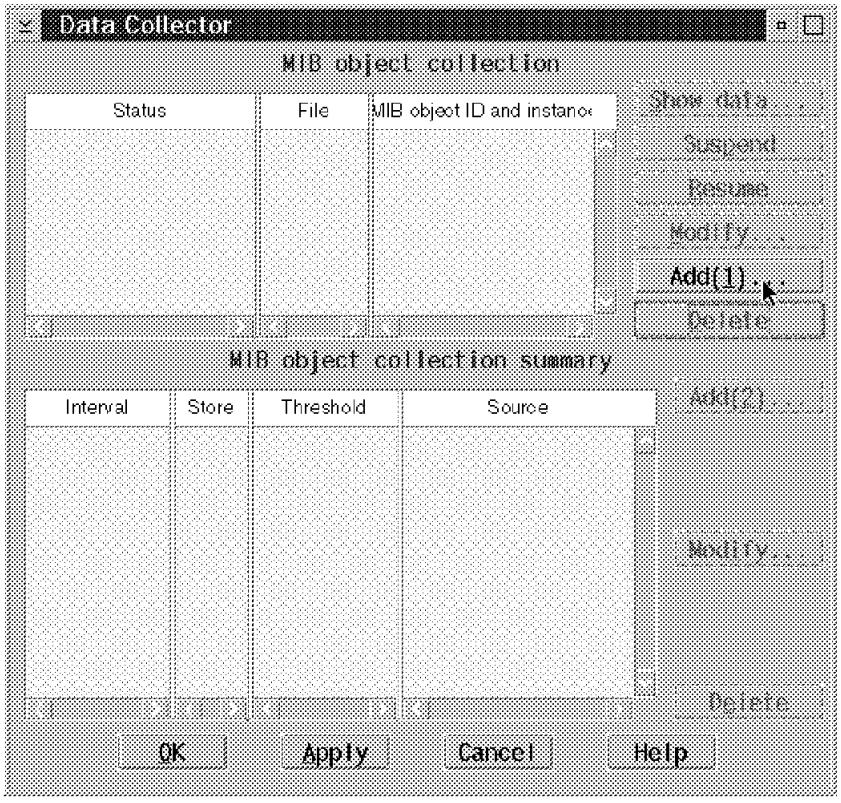


Figure 54. The Data Collector Screen

The only active button at this stage should be the Add(1) button. Click on this button so that we can add the MIB variable that we will collect data from.

We are going to monitor the CPU load of RALYAS4A and RALYAS4B using the MIB variable *nv6saComputerSystemLoad*. To get to this variable you need to move down the MIB tree.

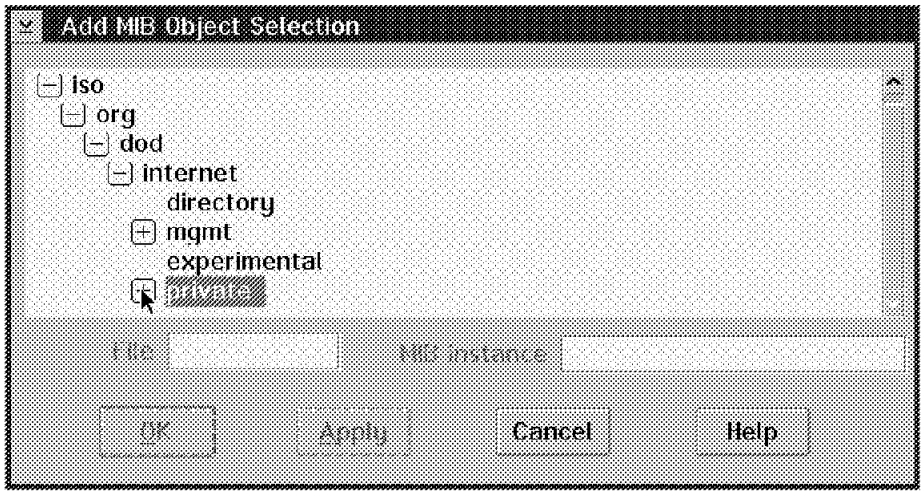


Figure 55. Selecting the MIB Variable to Monitor

Click on **private**.

Continue this process by clicking on the following objects until you arrive at the *nv6saComputerSystemLoad* object as shown in Figure 56.

enterprises→
ibm→
ibmProd→
netView6000SubAgent→
nv6saComputerSystem

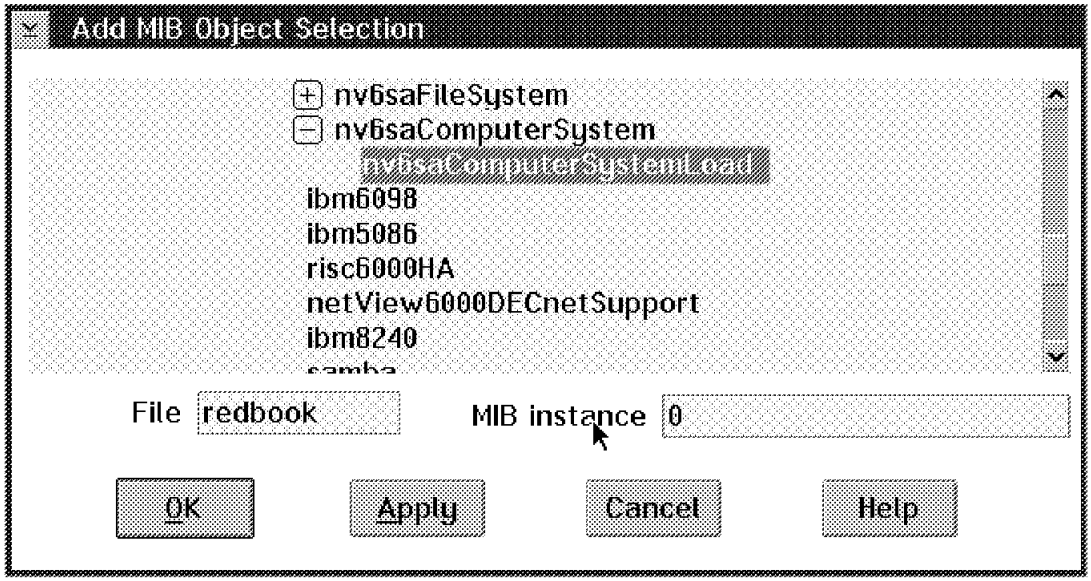


Figure 56. Specifying the File and MIB Instance

Specify the name of a file you would like the data collected to be stored in; we chose the name *redbook* but you should consider choosing a more meaningful name to make it easier to identify a file when you have multiple files.

Specify the MIB instance of the MIB variable. See 2.9.2.1, “Identification of Object Instances” on page 18 for a description of MIB object instances. Some MIB objects have several instances. Either type in the specific instance of the

object or use the wildcard character (*) to select all instances of the object. In this case there is only one instance, so we entered 0. Click on **OK**.

You will now see the Add Summary screen (see Figure 57).

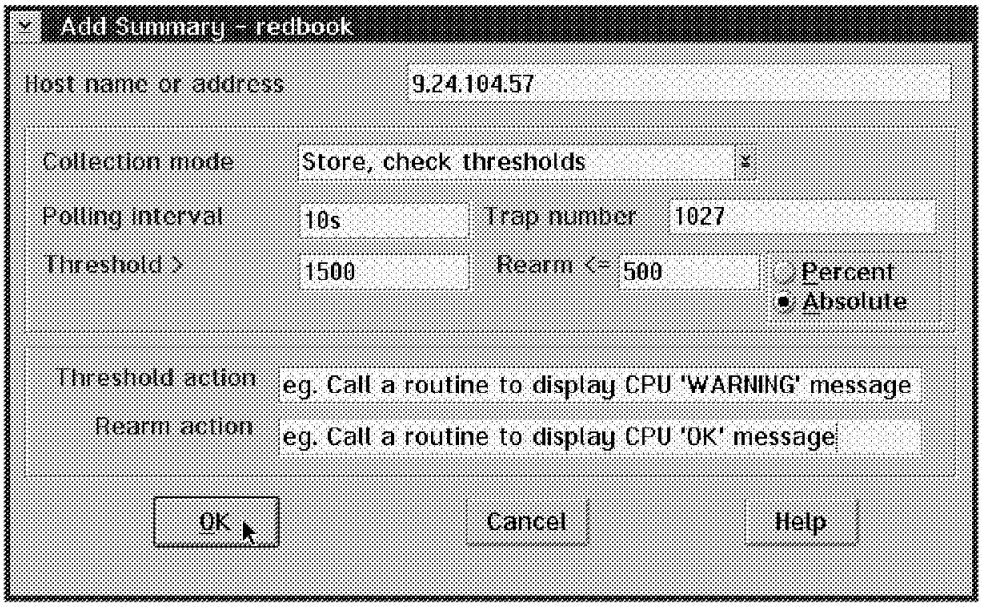


Figure 57. Adding a System to be Monitored

This is where we will add the address of the system from which we want to collect data. We can also specify whether to store the data or not and determine what to do if the value of the data exceeds certain thresholds we set on the screen.

Type the IP address of one of the systems to be monitored. In this case 9.24.104.57 (RALYAS4B).

Set the value of Collection mode to **Store, check thresholds**.

Set the polling interval. We chose 10 seconds.

Enter a user-defined trap number. This trap will be sent to your trap manager when the threshold conditions are met. We used trap number 1027.

Set the threshold to a value at which you would like a trap sent. We set it to 1500; this equates to 15% CPU load. So when the sampled value of this variable exceeds 1500, a trap will be sent to the trap manager with a trap number of 1027.

The Rearm value is used to *reset* the threshold trap and to signal, via another trap, when the sampled value of this MIB variable has gone below the Rearm value. We set it to 500 which equates to 5% CPU load. In our example a trap will be sent when CPU load exceeds 15% and another trap will be sent when the value drops below 5%. Subsequent threshold traps will not be sent until the rearm trap has been generated. Using these two values it is possible to call an OS/2 program or routine when either the threshold or rearm value has been reached.

Now click on **OK**. You should now see the Data Collector screen (Figure 58 on page 68) with your collection file and summary added. Using the Add(2) button

we will add another system to be monitored. In this example it is RALYAS4A at IP address 9.24.104.56.

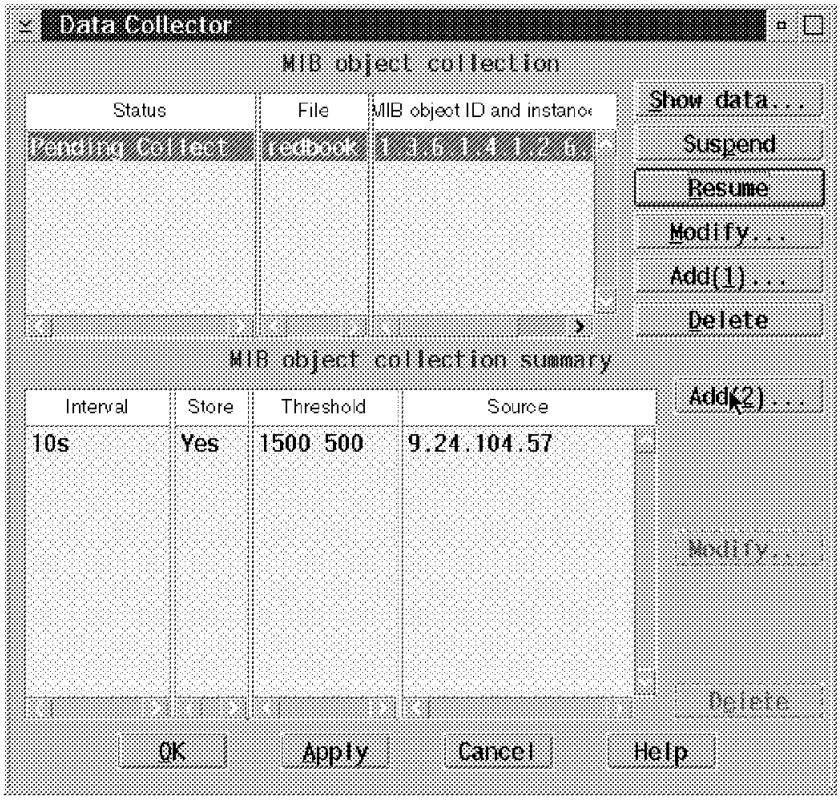


Figure 58. RALYAS4B Added

Repeat the procedure you just completed for the second system and click on the **OK** button when you have finished. You should now see both systems in the collection summary panel as in Figure 59 on page 69.

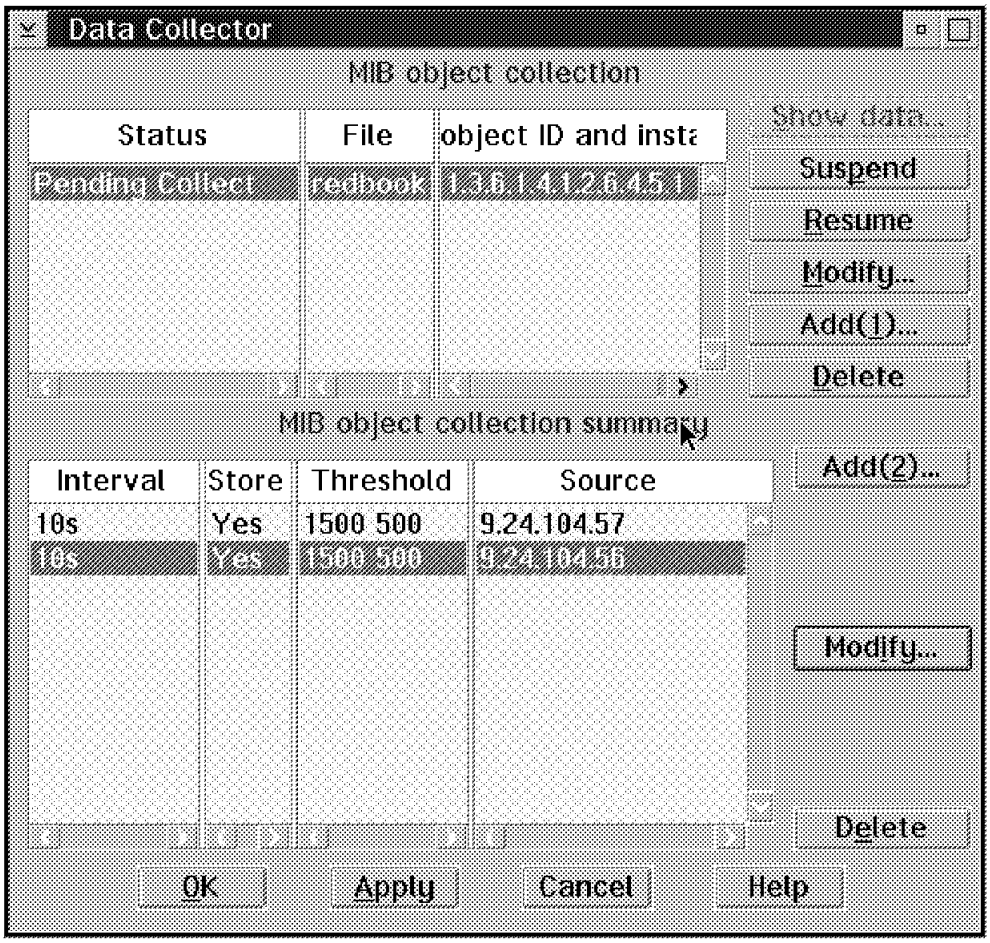


Figure 59. Two Systems Added

Now click on the **Resume** button to commence collection of data then click on the **Apply** button to set the values you have chosen. After the first collection interval has elapsed, the Show data button should be revealed. See Figure 60 on page 70.

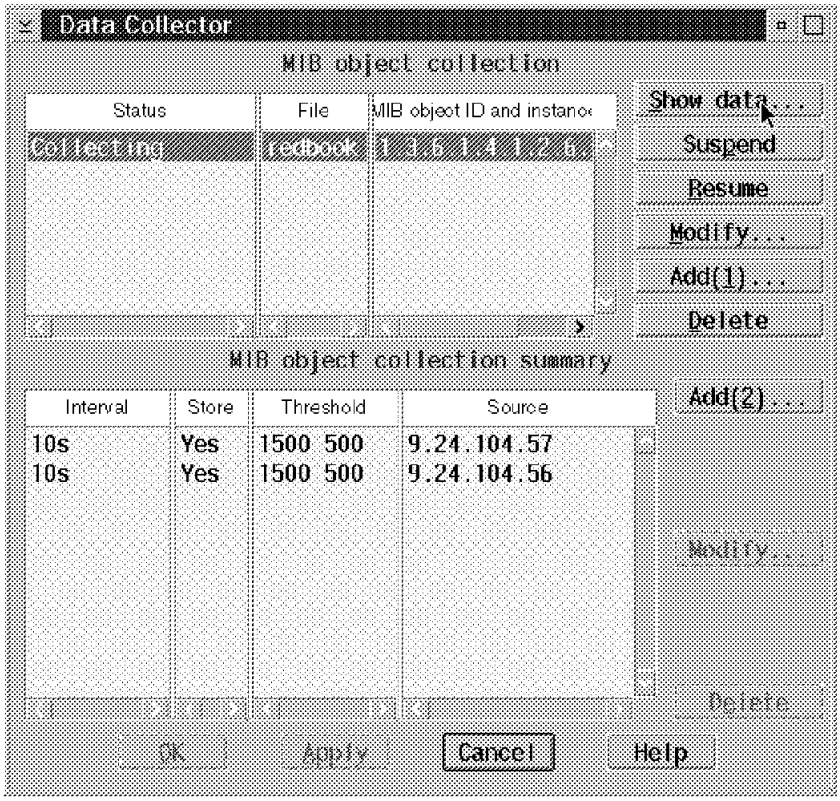


Figure 60. Commencing Data Collection

Click on the **Show data** button to view the data that has been collected since collection was resumed.

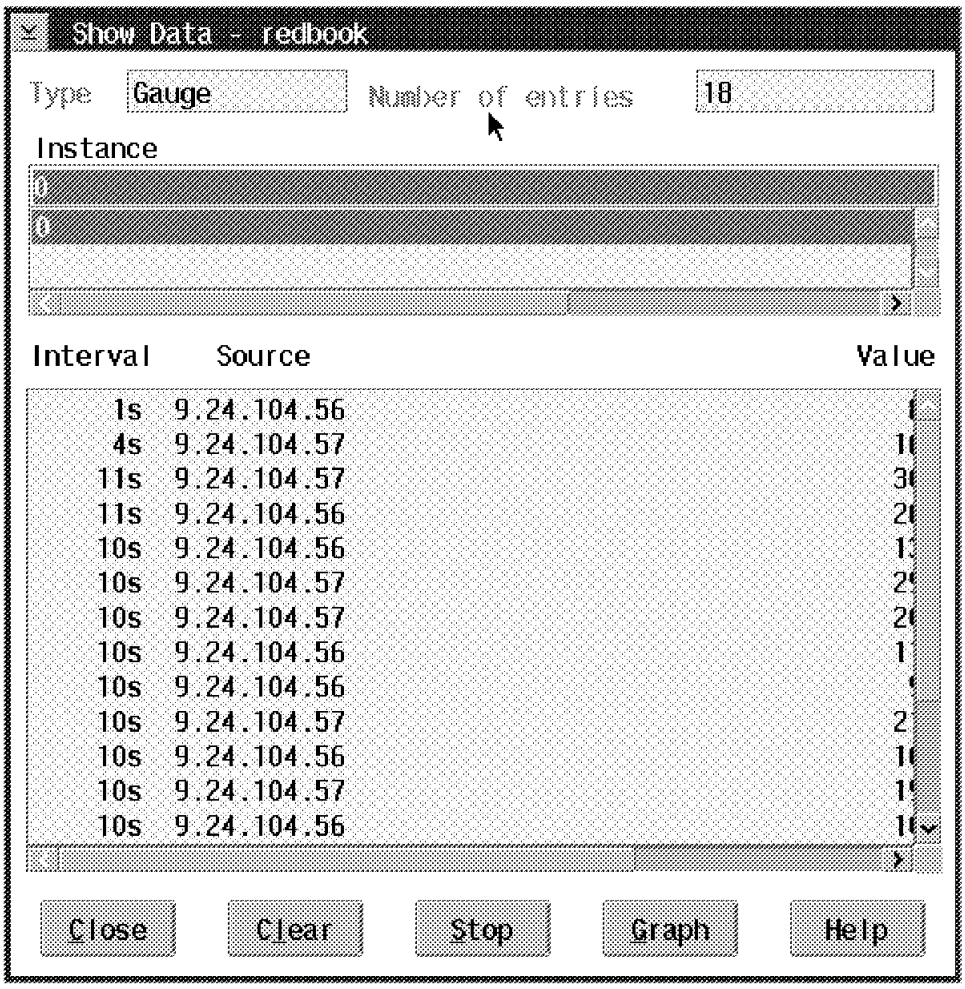


Figure 61. Collected Data

Note

The data collected is stored in a file in directory COLLECT. This directory is in the NetView for OS/2 path. In our case this is D:\anv2\collect (see Figure 62 on page 72).

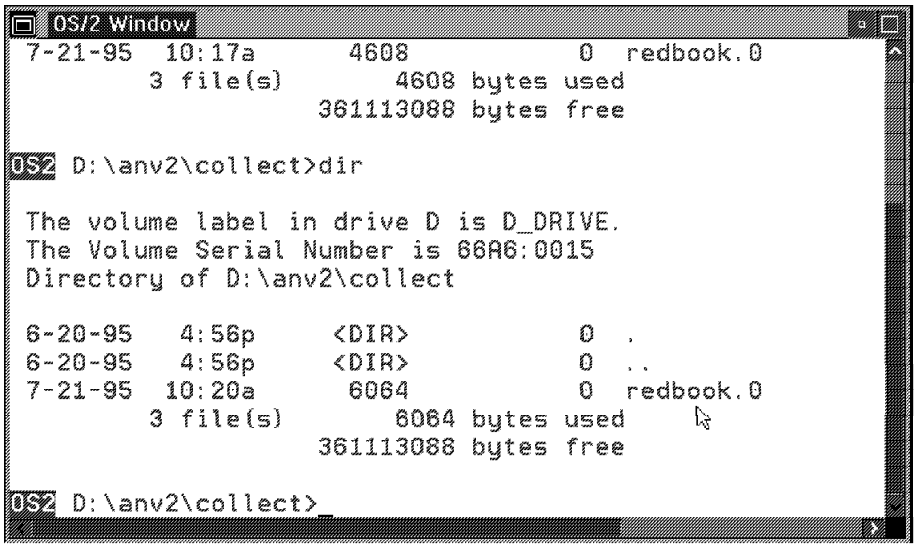


Figure 62. Collection Data Log File - Redbook

To display the data graphically, Click on the **Graph** button (see Figure 61 on page 71) to display the data collected so far (as shown in Figure 63).

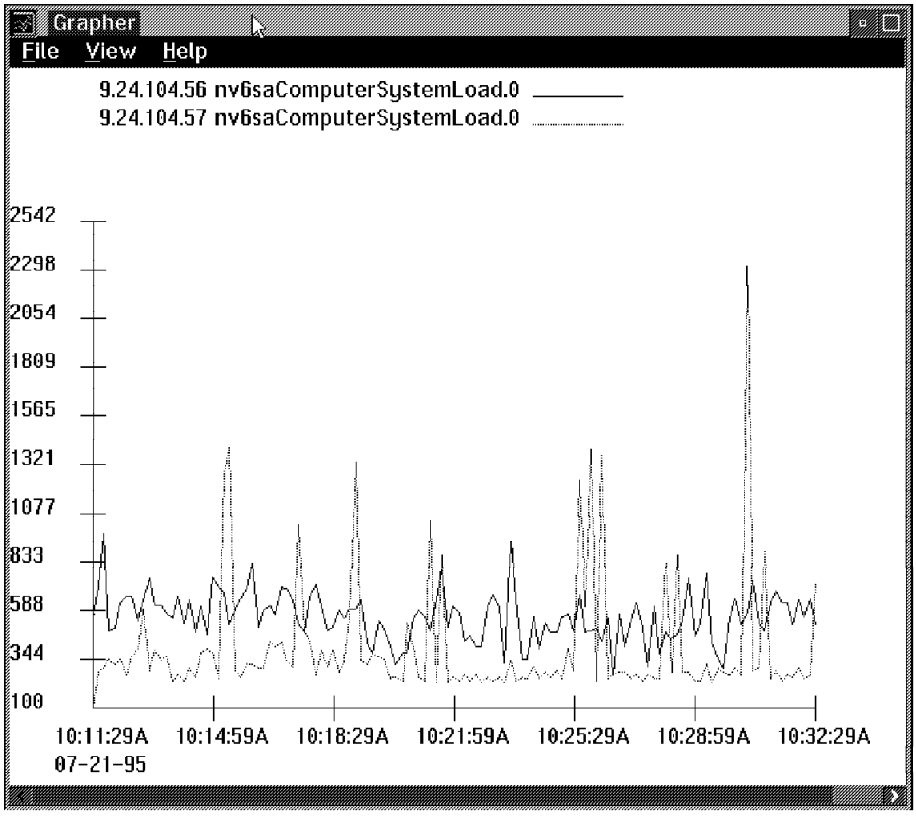


Figure 63. Graphical Display of Both Systems' CPU Load

Unlike the MIB Browser's graph function, this graph is not automatically updated at each polling interval. It just plots the data already logged.

We can now see quite easily how the two systems are performing. We can see the CPU load for RALYAS4A at IP address 9.24.104.56 briefly exceeded 15% at 10:28:59am and 10:32:29am. This should have generated our user-defined trap.

Because the NetView for OS/2 system is set up to send any traps to itself we can view the traps by clicking on the **Event Displayer** icon from the IBM NetView for OS2 folder.

If we specify a time range that includes a period of high CPU on the Events Display panel and click on the **Display** button, we see the All Events screen as shown in Figure 64.

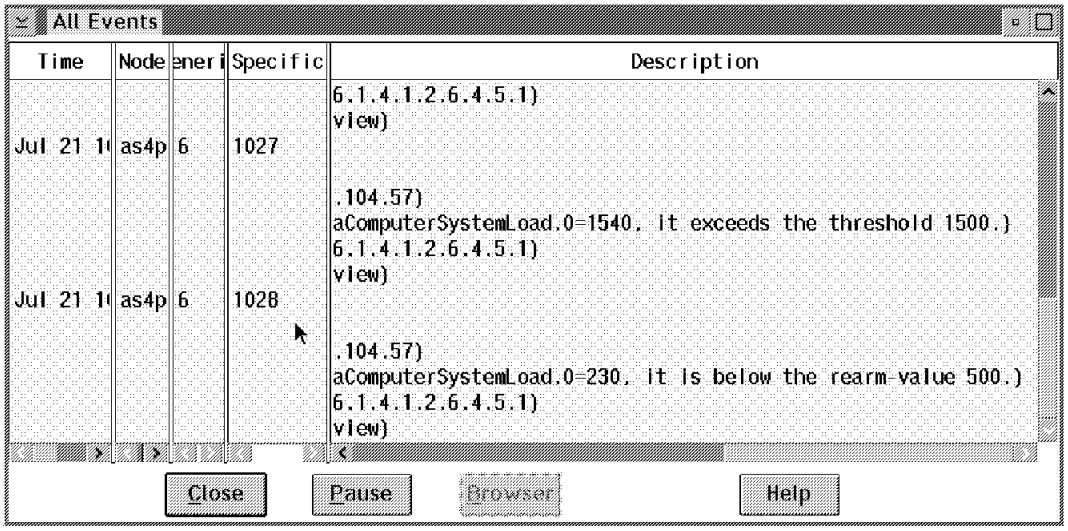


Figure 64. Displaying the Traps

You can see that two traps were generated; trap 1027 was generated when the CPU exceeded the value we set in Figure 57 on page 67 (15%), and trap 1028 was generated when it dropped below the rearm value (5%).

Using NetView for OS/2 to manage your AS/400s, it is possible to display pop-up messages when a specific trap is received or even dial someone's message pager and leave a textual message reporting the trap. This is done through the Event Handler function of NetView for OS/2.

Chapter 6. OS/400 SNMP Manager

This chapter describes how the SNMP manager is enabled for OS/400.

The OS/400 SNMP manager provides a rudimentary base for SNMP management applications. The OS/400 SNMP manager consists of the following functions:

- SNMP manager application program interface (API)
- Trap manager

For more OS/400 SNMP Manager information refer to the *System API Reference book*.

6.1 SNMP Manager APIs Overview

SNMP managing applications typically use APIs to establish communication with local or remote SNMP agents and then call other APIs to retrieve or modify MIB objects managed by those agents. The OS/400 SNMP manager APIs accomplish both of these tasks within the same API. Three manager APIs are provided to perform the SNMP GET, GETNEXT and SET operations. The communication mechanism between the manager APIs and the agents uses sockets. Therefore, both systems need to support sockets. SNMP uses the UDP protocol, therefore the manager APIs also use the UDP protocol. UDP is an unreliable protocol, the application builder must allow for this in the application design.

In general, all three APIs are blocked. That is, when the application calls these APIs, the API constructs a proper SNMP message (see Appendix A, “Additional Information on SNMP” on page 235 for more information on SNMP message layout), delivers it to the proper SNMP agent, waits, decodes the response from the agent, and delivers the information to the application. No processing occurs in the application until the API delivers this information or times out. See Figure 65 on page 76.

For additional information, see the Simple Network Management Protocol (SNMP) Manager APIs section of the *OS/400 System API Reference: UNIX Type APIs* book (SC41-3875 for V3R2 or SC41-4875 for V3R6).

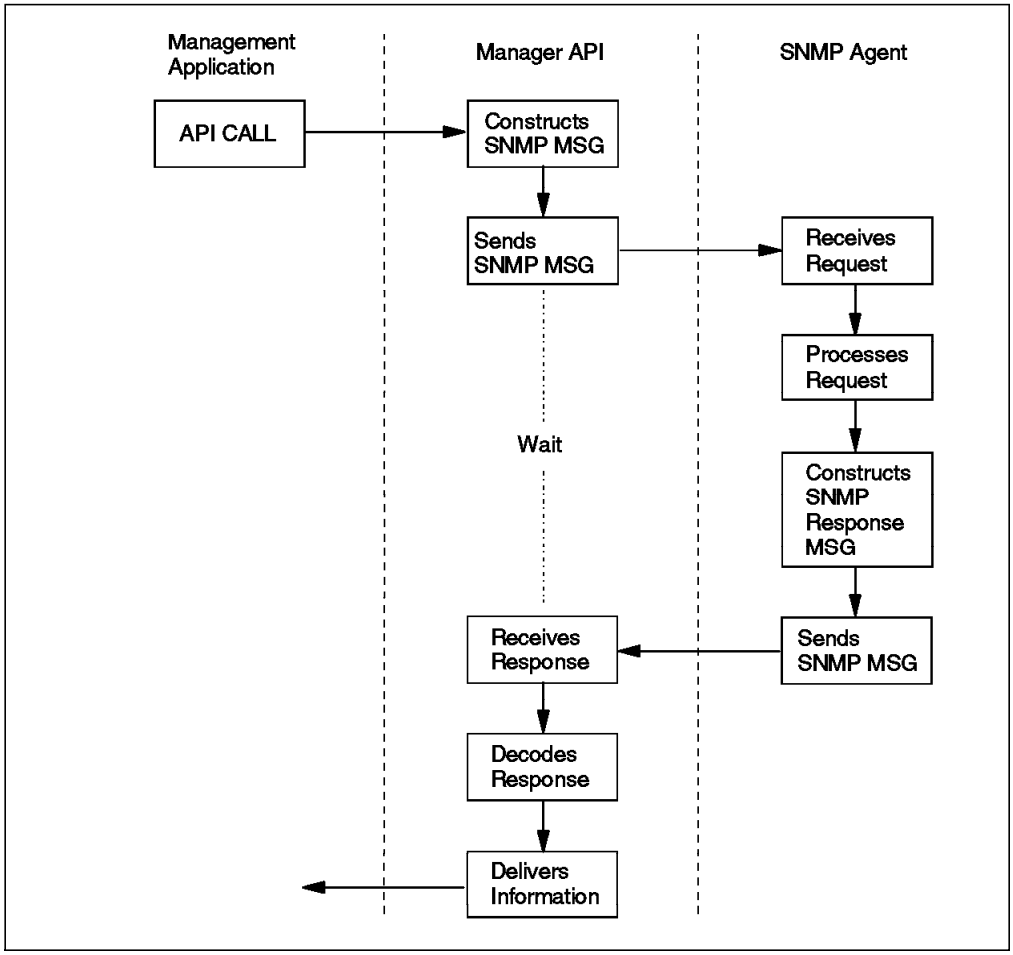


Figure 65. Manager API Overview

6.1.1 SNMP Manager API Functions

The OS/400 SNMP support provides the following three manager APIs:

snmpGet() This API is used by an SNMP managing application to get (retrieve) the value of one or more MIB objects from an SNMP agent or subagent on a local or remote system.

snmpGetnext() This API is used by an SNMP managing application to get (retrieve) the value of one or more MIB objects from an SNMP agent or subagent on a local or remote system. The snmpGetnext() function gets the value of the next available variable following the specified variable.

snmpSet() This API is used by an SNMP managing application to set (modify) the value of one or more MIB objects in an SNMP agent or subagent on a local or remote system.

For program development, the QSYSINC library must also be installed. QSYSINC library is installed via the Openness Includes OS/400 install option (option 13).

6.1.2 SNMP Manager Example

The following is an example of an API manager implementation.

This simple SNMP manager program does a ″MIB walk″ on an SNMP agent by doing repeated GETNEXT operations.

```
/* Necessary include files        */
#include <QTOMEAPI.H>
#include <stdio.h>
#include <string.h>
#include <milib.h>
#include <miproces.h>
#include <string.h>
#include <stdlib.h>

_MI_Time time_to_wait;
int hours = 0,
    minutes = 0,
    seconds = 5,
    hundreths = 0;

short wait_option = _WAIT_NORMAL;

snmppdu MGRpdu;
snmppdu *MGRpdu_p = &MGRpdu;
char MGRcommunity[6] = {0x70,0x75,0x62,0x6C,0x69,0x63}; /* public */
int rc;
char value[4000];
char oid[100]="1.3.6.1.2.1.1.3"; /* OID to begin at */
char loopback[10]="127.0.0.1"; /* loopback address */
char *addr_p = &loopback[0];

/* varbind */
varBind MGRvarbind;

void printValue(varBind *vb_p);

/* Main code starts here             */
int main(int argc, char *argv[]) {
if (argc > 1) {     /* If more than one argument is given   */
  /* Override default # of seconds to wait between requests */
  seconds = atoi(argv[1]);
}
if (argc > 2) {     /* If more than two arguments are given */
  /* Override default IP address */
  addr_p = argv[2];
}
mitime(&time_to_wait, hours, minutes, seconds, hundreths);

/* loop forever */
do {
  /* Init varbind */
  MGRvarbind.next = 0; /* ptr to next varbind */
  MGRvarbind.oid = &oid[0]; /* OID */
  MGRvarbind.asn_type = 0;
  MGRvarbind.val_len = 4000;
  MGRvarbind.val.str_val = 0;

  /* Create PDU */
```

```
  MGRpdu.varbind = &MGRvarbind;
  MGRvarbind.val.str_val=&value[0];

  /* MIB walk loop */
  do {
  /* GETNEXT */
  MGRvarbind.val_len=4000;
  rc = snmpGetnext(
          MGRpdu_p,
          addr_p,    /* agent IP address or host name */
          30,        /* time out in seconds */
          &MGRcommunity[0],
          6); /* community name length */
  if (rc == 0) { printValue(MGRpdu.varbind); }
  else { printf("rc=%d\n",rc); }  /* print the return code */
  waittime(&time_to_wait, wait_option);
} while (rc == 0);

} while (1); /* forever */

}

void printValue(varBind *vb_p) {
  printf("%s: ",vb_p->oid);
  switch (vb_p->asn_type) {
    case (API_ASN_OCTET_STRING):
    case (API_ASN_OBJECT_IDENTIFIER):
      *((vb_p->val.str_val)+vb_p->val_len)=0;
      /* convert ASCII to EBCDIC */
      /* printf("%s",*(vb_p->val.str_val)); */
      break;
    default:
      printf("%d",*(vb_p->val.int_val));
      break;
  }
  printf("\n");
}
```

6.2 Trap Manager Overview

The OS/400 SNMP trap support provides the user the capability of monitoring for unsolicited SNMP trap messages. These trap messages may contain helpful information for managing a network.

The OS/400 SNMP manager has the following two trap functions:

- Forward a received trap to other SNMP managers
- Deliver a received trap to a data queue

Figure 66 on page 79 provides an overview of the AS/400 SNMP trap manager functions.

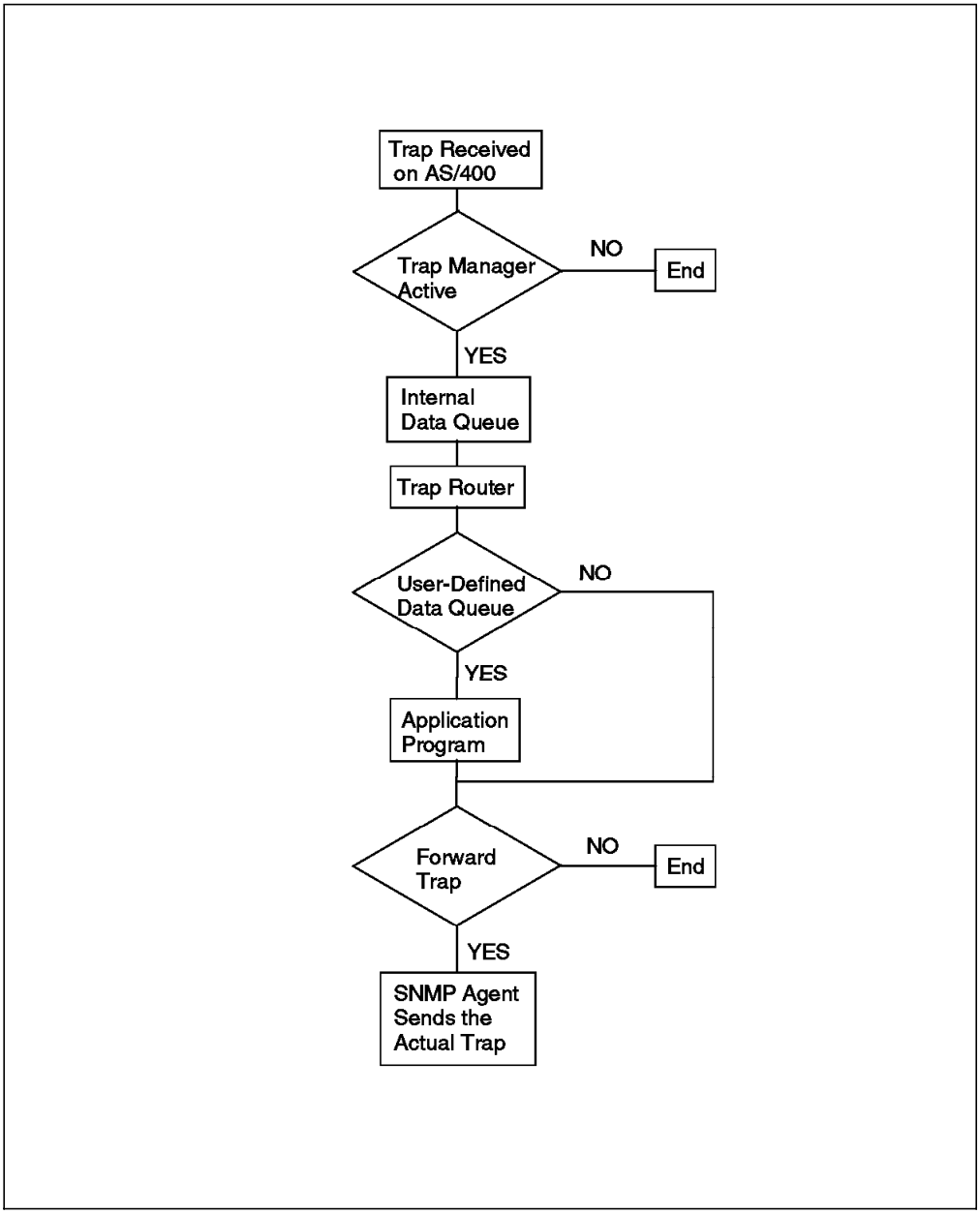


Figure 66. Trap Manager Overview

The trap manager receives, parses and then enqueues traps received to an internal data queue. All traps that are received on an AS/400 system can also be routed to user-defined data queues. See 6.2.2.1, “Configuring Trap Support for Data Queue Delivery” on page 84 for more information on how to specify a user-defined data queue.

If configured to do so, the trap manager then forwards the traps to other SNMP management destinations as configured in the SNMP attributes. The traps are forwarded through the SNMP agent that generates and sends the actual trap PDU. Therefore, traps are sent to all managers that are configured in the SNMP agent’s trap manager attributes. See 6.2.1, “Trap Forwarding” on page 80 for more information on how to forward traps.

The commands to start and end the OS/400 trap manager support are the following:

- STRTRPMGR - Start Trap Manager
- ENDTRPMGR - End Trap Manager

The STRTRPMGR command starts the following jobs:

- QTRAPMGR
- QTRAPRCV

The WRKACTJOB command can be used to verify that these jobs are running under QSYSWRK subsystem. The SNMP agent must be active if trap forwarding is desired.

Use the command WRKACTJOB SBS(QSYSWRK) as in Figure 67 to display the trap manager jobs.

```
                      Work with Active Jobs            RALYAS4A
                                                            08/01/95
CPU %:     .0     Elapsed time:   00:00:00     Active jobs:   66

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...

Opt Subsystem/Job User        Type  CPU %  Function        Status
    QSYSWRK       QSYS        SBS      .0                 DEQW
     QAPPCTCP     QSYS        BCH      .0  PGM-QZPAIJOB    TIMW
     QCQEPMON     QSVMSS      BCH      .0  PGM-QCQEPMON    MSGW
     QCQRCVDS     QSVMSS      BCH      .0  PGM-QCQAPDRM    MSGW
     QECS         QSVSM       BCH      .0  PGM-QNSECSJB    DEQW
     QMSF         QMSF        BCH      .0                 DEQW
     QTMSNMPRCV   QTCP        BCH      .0  PGM-QTOSRCVR    TIMW
     QTRAPMGR     QTCP        BCH      .0  PGM-QTOMMAIN    DEQW
     QTCPIP       QTCP        BCH      .0                 DEQW
                                                             More...
 Parameters or command
 ===>
 F3=Exit    F5=Refresh  F10=Restart statistics  F11=Display elapsed time
 F12=Cancel   F23=More options   F24=More keys
```

Figure 67. Trap Manager Jobs Running Under QSYSWRK

6.2.1 Trap Forwarding

The Start Trap Manager (STRTRPMGR) command should be used to start the OS/400 SNMP trap manager job. This command has an optional forward trap parameter. This parameter enables all traps received on port 161 of this system to be forwarded to the SNMP managers listed in the SNMP's agent trap manager attribute. Port 161 is the *well-known* port for SNMP.

The CHGSNMPA command should be used to specify the internet address and the community of those managers the traps should be forwarded to. Up to 300 managers can be specified in the the Trap Manager (TRPMGR) parameter.

Note: Be careful when configuring the trap destinators to avoid trap loops. Loops are caused when two or more trap managers are configured to forward traps to each other.

Traps will only be forwarded if the trap manager job is started specifying forward *YES in the FWDTRP parameter.

```
STRTRPMGR FWDTRP(*YES)
```

6.2.1.1 Configuring Trap Support for Trap Forwarding

The following is an example of how traps are forwarded to different systems. We had the following environment:

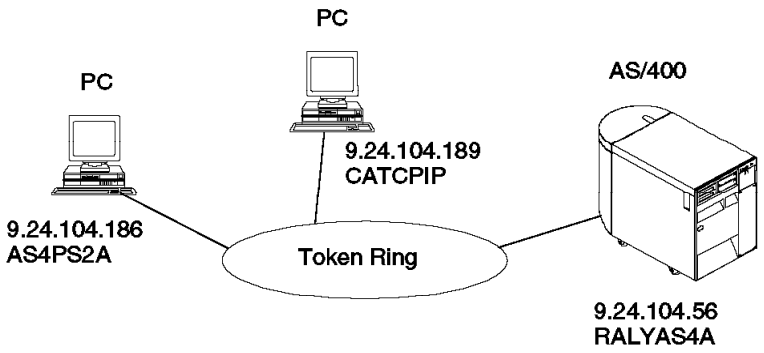


Figure 68. Lab Environment

When the system CATCPIP (configured with the SNMP support provided with Client Access/400 Optimized for OS/2) starts the SNMP agent, which is part of Client Access, it will send a *coldStart* trap to RALYAS4A (as configured in Client Access). System RALYAS4A will be configured to forward the trap to system AS4PS2A (our NetView for OS/2 system).

RALYAS4A: The following should be configured at system RALYAS4A:

- Specify the internet address and community name of the manager the traps will be forwarded to, using the following command:

  ```
  CHGSNMPA (Press F4)
  ```

```
                      Change SNMP Attributes (CHGSNMPA)

Type choices, press Enter.

System contact . . . . . . . . . > 'Maria Hermelo, Lee Hargreaves & Fra
cisco Infante - Tele 123 456 7890'


System location  . . . . . . . . > 'Raleigh ITSO - North Carolina
RALYAS4A'


Send authentication traps  . . .   *YES     *SAME, *YES, *NO
Automatic start  . . . . . . . . > *NO      *SAME, *YES, *NO
Object access  . . . . . . . . .   *READ    *SAME, *READ, *WRITE, *NONE
Log set requests . . . . . . . . > *NO      *SAME, *YES, *NO
Log get requests . . . . . . . . > *NO      *SAME, *YES, *NO
Log traps  . . . . . . . . . . .   *YES     *SAME, *YES, *NO

                                                                More...

F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

```
                      Change SNMP Attributes (CHGSNMPA)

Type choices, press Enter.

Trap managers:
  Manager internet address . . . > '9.24.104.186'
  Community name . . . . . . . . > 'public'



  Translate community name . . . > *YES          *YES, *NO
              + for more values









                                                              Bottom

F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

Figure 69. Forwarding Traps to System AS4PS2A

The IP address 9.24.104.186 specified in the TRPMGR parameter indicates that traps should be forwarded to system AS4PS2A. The community name *public* is a valid community name for RALYAS4A and AS4PS2A.

The Log Traps (LOGTRP) parameter is set to *YES (to log all the traps received).

- The trap manager must be started specifying *YES in the FWDTRP parameter using the following command: STRTRPMGR FWDTRP(*YES).

6.2.1.2 Generating and Forwarding a Trap

When Client Access/400 Optimized for OS/2 SNMP support is started on CATCPIP, it will send a *coldStart* trap to RALYAS4A (as configured in SNMP support for Client Access).

Using the DSPJRN (QUSRSYS/QSNMP) command, if *YES was specified in the LOGTRP parameter of the SNMP agent attributes, the following entry is logged in the QSNMP journal of system RALYAS4A.

```
                          Display Journal Entry

Object . . . . . :                  Library  . . . . . :
Member . . . . . :                  Sequence . . . . . :   10870
Code . . . . . :   M  - Network management data
Type . . . . . :   SN - SNMP information

            Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00001      'T                     9 241041860   00   0       '
00051      '                                                 '
00101      '                 '
```

Figure 70. QSNMP Journal Entry for System RALYAS4A

This journal entry indicates that a trap was sent to a trap manager at internet address 9.24.104.186 (AS4PS2A). There is no community name specified, as this system (RALYAS4A) is just delivering to system AS4PS2A the trap received from CATCPIP. Field 39 indicates that the trap type is *coldStart* (0). Error type and error index fields are both zero. For more information on journal entries, refer to Chapter 12, “Journal for SNMP Logging” on page 211. For more information on trap types, refer to A.3.2.5, “The Trap-PDU” on page 243.

Figure 71 shows the trap received at AS4PS2A (the NetView for OS/2 system).

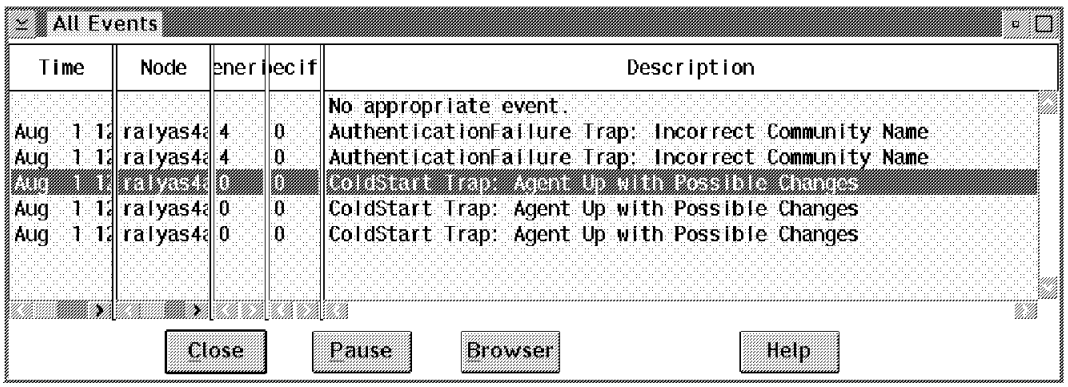


Figure 71. Trap Logging Facility of NetView for OS/2

In the Node field the name of the system forwarding the trap can be seen. In this case it is system RALYAS4A. The Generic field indicates the kind of trap received. The zero (0) indicates that a *coldStart* trap was received. In the Description field ″ColdStart Trap: Agent Up with possible Changes″ can be seen.

Through this example we have shown how to configure the systems for trap forwarding and how to monitor them. You can also monitor traps with the use of user-defined data queues.

6.2.2 Traps Delivered to Data Queues

Using the OS/400 SNMP manager, it is possible to deliver SNMP traps to data queues. All traps that are received on an AS/400 system can be routed to user-defined data queues. An application should monitor the data queue to receive trap information. Depending on the kind of trap received in the data queue, the application can take an action, such as, notifying the system operator.

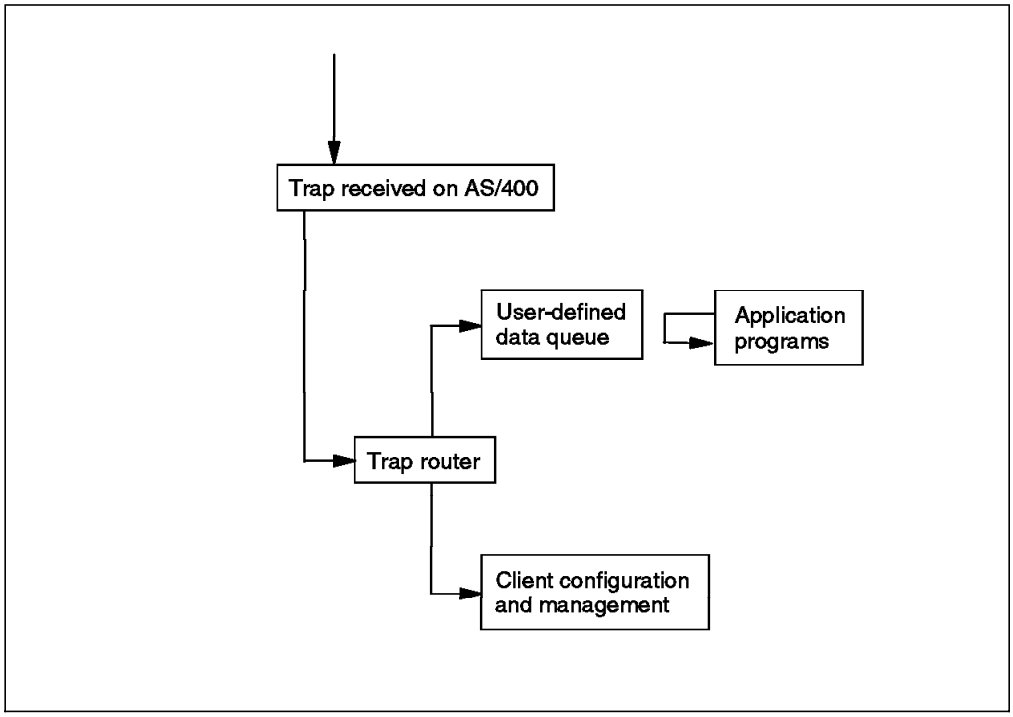


Figure 72. SNMP Trap Support

6.2.2.1 Configuring Trap Support for Data Queue Delivery

The SNMP trap support must be started to deliver traps to data queues using the STRTRPMGR command. If the trap manager is started, traps will always be delivered to an internal data queue, and to a user-defined data queue, if specified. The FWDTRP parameter of the STRTRPMGR command has no effect on the delivery to a data queue.

To define a user data queue to receive the traps, the data queue name has to be specified in the SNMP trap support exit point. SNMP trap support uses the exit point QIBM_QZCA_SNMPTRAP and a data queue you define. Up to 100 data queues can be specified.

The following steps are required to configure the OS/400 SNMP trap support to deliver traps received to a data queue:

1. Use the Work with Registration Information (WRKREGINF) command (as follows) to determine if the QIBM_QZCA_SNMPTRAP exit point exists on your system:

   Enter the command WRKREGINF.

```
                      Work with Registration Information

Type options, press Enter.
  5=Display exit point   8=Work with exit programs

                         Exit
    Exit                 Point
Opt Point                Format   Registered  Text
    QIBM_QRQ_SQL         RSQL0100    *YES     Original Remote SQL Server
    QIBM_QTF_TRANSFER    TRAN0100    *YES     Original File Transfer Fun
    QIBM_QVP_PRINTERS    PRNT0100    *YES     Original Virtual Print Ser
    QIBM_QZCA_ADDC       ZCAA0100    *YES     Add Client exit point
    QIBM_QZCA_REFC       ZCAF0100    *YES     Refresh Client Information
    QIBM_QZCA_RMVC       ZCAR0100    *YES     Remove Client exit point
    QIBM_QZCA_SNMPTRAP   ZCAT0100    *YES
    QIBM_QZCA_UPDC       ZCAU0100    *YES     Update Client Information
    QIBM_QZDA_INIT       ZDAI0100    *YES     Database Server - entry
    QIBM_QZDA_NDB1       ZDAD0100    *YES     Database Server - data bas
    QIBM_QZDA_NDB1       ZDAD0200    *YES     Database Server - data bas
                                                                  More..
Command
===>
F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
```

Figure 73. Registered Exit Points

2. If the exit point QIBM_QZCA_SNMPTRAP does not exist, the exit point should be created and registered using the following command:

   ```
   CALL PGM(QUSRGPT) PARM('QIBM_QZCA_SNMPTRAP ' 'ZCAT0100' X'00000000'
   X'00000000')
   ```

   Note: The first parameter *must* be 20 characters long.

3. A data queue of 32780 bytes has to be defined. For example, to define a data queue called TRAPQ in library QGPL, enter the following command:

   ```
   CRTDTAQ DTAQ(QGPL/TRAPQ) MAXLEN(32780)
   ```

4. The following should be done to register the exit program and the exit program data with QIBM_QZCA_SNMPTRAP exit point:

   a. Enter the command WRKREGINF.

```
                      Work with Registration Information

Type options, press Enter.
  5=Display exit point   8=Work with exit programs

                         Exit
    Exit                 Point
Opt Point                Format   Registered  Text
    QIBM_QRQ_SQL         RSQL0100    *YES     Original Remote SQL Server
    QIBM_QTF_TRANSFER    TRAN0100    *YES     Original File Transfer Fun
    QIBM_QVP_PRINTERS    PRNT0100    *YES     Original Virtual Print Ser
    QIBM_QZCA_ADDC       ZCAA0100    *YES     Add Client exit point
    QIBM_QZCA_REFC       ZCAF0100    *YES     Refresh Client Information
    QIBM_QZCA_RMVC       ZCAR0100    *YES     Remove Client exit point
 8  QIBM_QZCA_SNMPTRAP   ZCAT0100    *YES
    QIBM_QZCA_UPDC       ZCAU0100    *YES     Update Client Information
    QIBM_QZDA_INIT       ZDAI0100    *YES     Database Server - entry
    QIBM_QZDA_NDB1       ZDAD0100    *YES     Database Server - data bas
    QIBM_QZDA_NDB1       ZDAD0200    *YES     Database Server - data bas
                                                                  More..
Command
===>
F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
```

Figure 74. Working with Registration Information

b. From the Work with Registration Information panel take option 8 (Work with exit programs) for the QIBM_QZCA_SNMPTRAP exit point and press Enter.

```
                            Work with Exit Programs

Exit point:   QIBM_QZCA_SNMPTRAP       Format:   ZCAT0100

Type options, press Enter.
  1=Add   4=Remove   5=Display   10=Replace

              Exit
            Program     Exit
Opt          Number     Program        Library
 1

  (No exit programs found.)









                                                                  Bottom
Command
===>
F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F12=Cancel
```

Figure 75. Working with Exit Programs

c. From the Work with Exit Programs panel take option 1 (Add) and then press F10 (additional parameters). The panel as shown in Figure 76 on page 87 will be presented.

```
                        Add Exit Program (ADDEXITPGM)

Type choices, press Enter.

Exit point . . . . . > QIBM_QZCA_SNMPTRAP
Exit point format  . . . . . > ZCAT0100     Name
Program number . . . . . > 1            1-2147483647, *LOW, *HIGH
Program  . . . . . .   TRAPPGM      Name
  Library  . . . . . .     QGPL       Name, *CURLIB
Text 'description' . . . . . .   *BLANK


                          Additional Parameters

Replace existing entry . . . . > *NO          *YES, *NO
Create exit point  . . . . . .   *NO          *YES, *NO




                                                                    More...
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

```
                        Add Exit Program (ADDEXITPGM)

Type choices, press Enter.

Exit program data:
  Coded character set ID . . . .   *NONE         Number, *NONE, *JOB
  Length of data . . . . . .                     0-2048, *CALC
  Program data . . . . . .   TRAPQ

                                                                      ...









                                                                  Bottom
F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this display
F24=More keys
```

Figure 76. Adding an Exit Program

d. Enter the name of the program that will monitor the data queue to receive the trap information. In this example the program is TRAPPGM in library QGPL.

e. Enter the name of the data queue created to receive the traps in the Program data field. In this example the data queue is called TRAPQ.

Note: This configuration only registers the data queue name. The program name and library that will use this data queue may be specified, even though this information is not used.

Notes:

1. The program data field on the Add Exit Program (ADDEXITPGM) display (see Figure 76 on page 87) contains the library name and the data queue name that will be used by the trap manager. This data must not exceed 21 bytes.
2. The exit program and library do not have to exist when they are added to the exit point.
3. The library name and data queue must be specified in *uppercase* on the exit point.
4. Multiple exit points are supported on the QIBM_QZCA_SNMPTRAP exit point. Each exit program must contain only one data queue.
5. A maximum of a hundred data queues can be defined.
6. The data queue names are retrieved from the exit point only when the trap manager is started. To activate any changes to the data queues, you must end the trap manager with the End Trap Manager (ENDTRPMGR) command and restart the trap manager with the Start Trap Manager (STRTRPMGR) command.
7. In the preceding scenario, all traps are added to the data queue. If the data queue is locked, damaged, destroyed, or named incorrectly, the traps are lost. It is the responsibility of the user to remove traps from the queue. No messages are sent if the queue is full or traps not removed.
8. The format for the ZCAT0100 trap data queue entry follows. For details about the SNMP trap, refer to the Internet standard for the trap message described in RFC1155 or RFC1157. The values for the object type field can be found in the AS/400 library QSYSINC, file H, member QTOMEAPI.

Table 1 (Page 1 of 2). Format for ZCAT0100 Trap Data Queue Entry

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Entry type (always *SNMPTRAP)
10	A	CHAR(2)	Entry ID (currently 01)
12	C	BINARY(4)	Version (This is the start of the trap header. All displacements are from the start of the trap header).
16	10	BINARY(4)	Length of community name
20	14	BINARY(4)	Displacement to community name
24	18	BINARY(4)	Length of enterprise object ID
28	1C	BINARY(4)	Displacement of enterprise object ID
32	20	BINARY(4)	Length of agent address
36	24	BINARY(4)	Displacement of agent address
40	28	BINARY(4)	Generic trap type
44	2C	BINARY(4)	Specific trap code
48	30	BINARY(4)	Time stamp
52	34	BINARY(4)	Number of variable bindings

Table 1 (Page 2 of 2). Format for ZCAT0100 Trap Data Queue Entry

Offset			
Dec	**Hex**	**Type**	**Field**
56	38	BINARY(4)	Displacement to first variable binding
Note: An array of variable bindings follows.			
This fields repeat for each variable binding		BINARY(4)	Length of object name
		BINARY(4)	Displacement of object name
		BINARY(4)	Length of value
		BINARY(4)	Displacement to value
		BINARY(4)	Value type (Values for this field can be found in AS/400 library QSYSINC, file H, member QTOMEAPI)
Note: All object names and values follow.			
		CHAR(*)	Object names and values for all variable bindings.

Chapter 7. OS/400 SNMP Subagent

This chapter describes the SNMP DPI routines supported by OS/400 and the Application Programming Interface (API) available in OS/400 for constructing an SNMP subagent.

For more OS/400 SNMP Subagent information refer to the Simple Network Management Protocol (SNMP) Subagent APIs section of the *OS/400 System API Reference: UNIX Type APIs* book (SC41-3875 for V3R2 or SC41-4875 for V3R6).

Further information on DPI programming, sample subagent code and documentation is available via anonymous FTP from:

```
software.watson.ibm.com
```

To obtain the source, perform the following steps:

```
ftp software.watson.ibm.com
user: anonymous
password: your e-mail address
cd /public/dpi
get README
binary
get dpi_api.tar (or compressed dpi_api.tar.Z)
quit
```

7.1 Introduction to SNMP Distributed Protocol Interface

The Simple Network Management Protocol agent distributed protocol interface (DPI) permits users to dynamically add, delete, or replace management variables in the local Management Information Base (MIB) without requiring you to recompile the SNMP agent.

For additional information refer to RFC1592 (SNMP DPI 2.0 RFC). See B.1.1.1, “How to Obtain a Copy of an RFC” on page 246 for information on how to obtain RFCs.

7.2 SNMP Agent and Subagents

SNMP agents are responsible for answering SNMP requests from network management stations (SNMP managers). These requests can be GET, GETNEXT and SET, performed on the MIB objects.

A subagent extends the set of MIB objects provided by the SNMP agent. An SNMP subagent is the implementation of a network management application on a managed system, which interfaces with the SNMP agent for the purpose of expanding the number of MIB objects that an SNMP manager can access. A subagent allows the addition of other MIB objects and registers them with the SNMP agent without the need to change or recompile the agent. Whether a MIB object is referenced by the agent or the subagent, is transparent to the SNMP manager.

When the agent receives a request for a MIB variable that is not a request for a MIB-II variable, it passes the request to the subagent. The subagent gathers the requested information and sends a response to the agent. The agent creates an

SNMP response packet and sends the response to the remote network management station that initiated the request. The existence of the subagent is transparent to the network management station.

The only MIB that is implemented within OS/400 SNMP agent is MIB-II. The other MIBs supported by the OS/400 SNMP such as APPN MIB, Client Management Inventory and the NetView/6000 Subagent MIB are all implemented as OS/400 SNMP subagents.

If there is some information that is required to manage a network and this information is not provided by the MIBs supported by the agent, a subagent can be built to provide this information without any kind of change having to be done in the SNMP agent.

7.3 DPI Agent Requests

In order to implement a subagent there are some requests that are initiated by the SNMP agent; others are initiated by the SNMP subagent and others are used by the SNMP subagent to communicate with the SNMP agent. Following is the list of requests and operations needed to build an SNMP subagent:

- The SNMP agent can initiate the following DPI requests:
  - GET
  - GETNEXT
  - SET, COMMIT and UNDO
  - UNREGISTER
  - CLOSE
- The SNMP subagent can initiate the following DPI requests:
  - OPEN, CLOSE
  - REGISTER, UNREGISTER
  - RESPONSE
  - TRAP
- The SNMP subagent communicates with the SNMP agent using the following functions:
  - CONNECT, DISCONNECT
  - RECEIVE, WAIT
  - SEND

The following charts show a sequence of events starting with the connection establishment between the subagent and agent through to the closing of the connection. The flow charts are intended to help in the understanding of SNMP subagent programming.

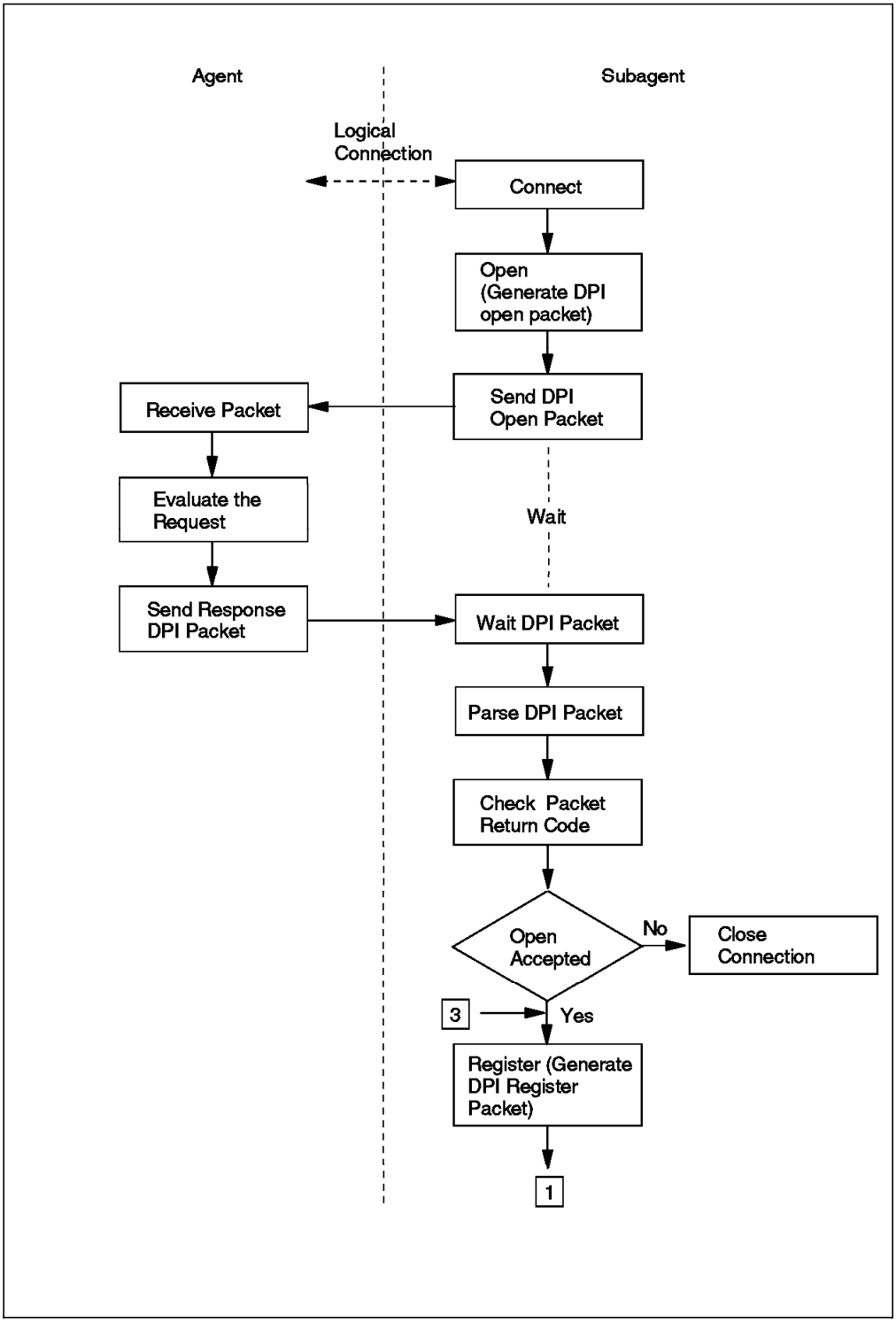


Figure 77. SNMP Subagent Connection and Register (Part 1)

Figure 77 and Figure 78 on page 94 show the initial connection establishment and registration between the subagent and agent.

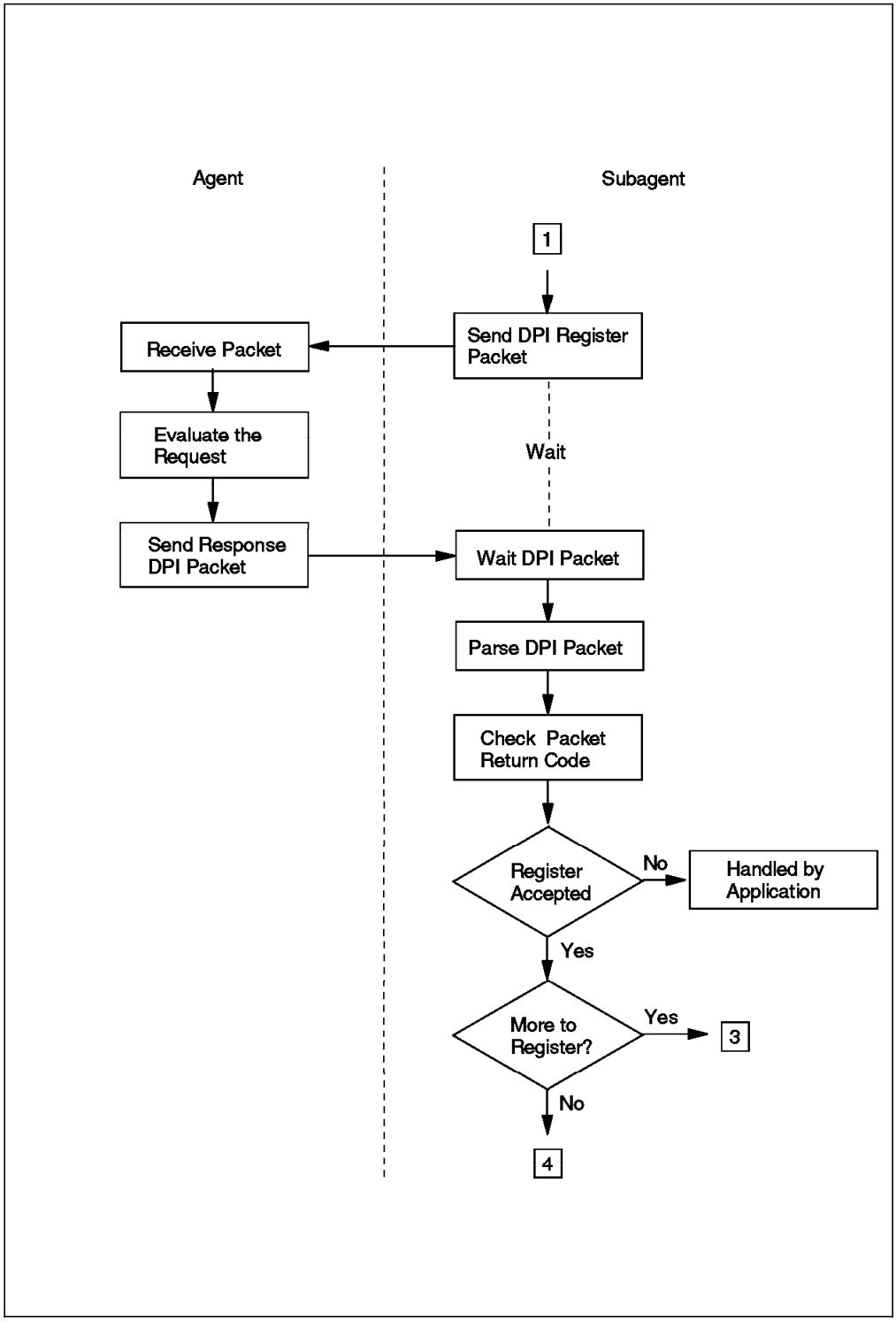


Figure 78. SNMP Subagent Connection and Register (Part 2)

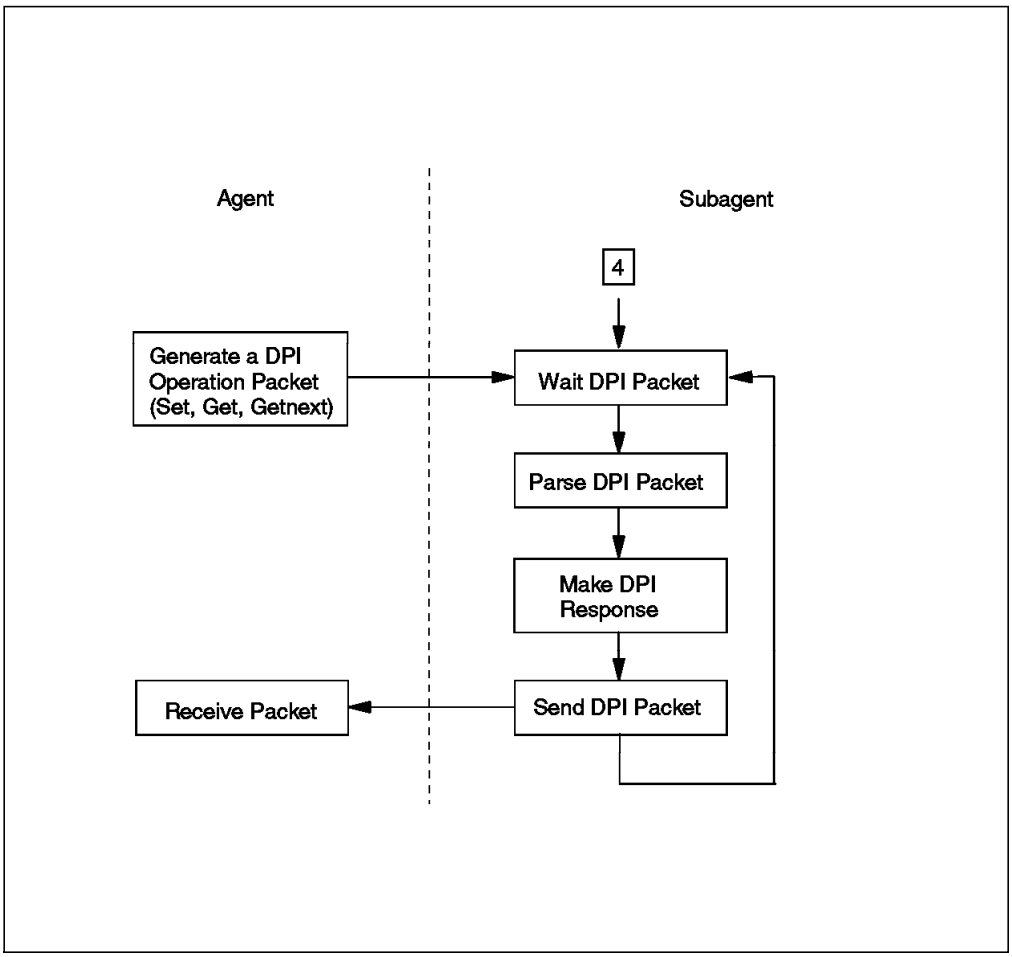


Figure 79. SNMP Subagent Wait and Execute Request from SNMP Agent

Figure 79 shows the SNMP subagent waiting and then executing requests from the SNMP agent.

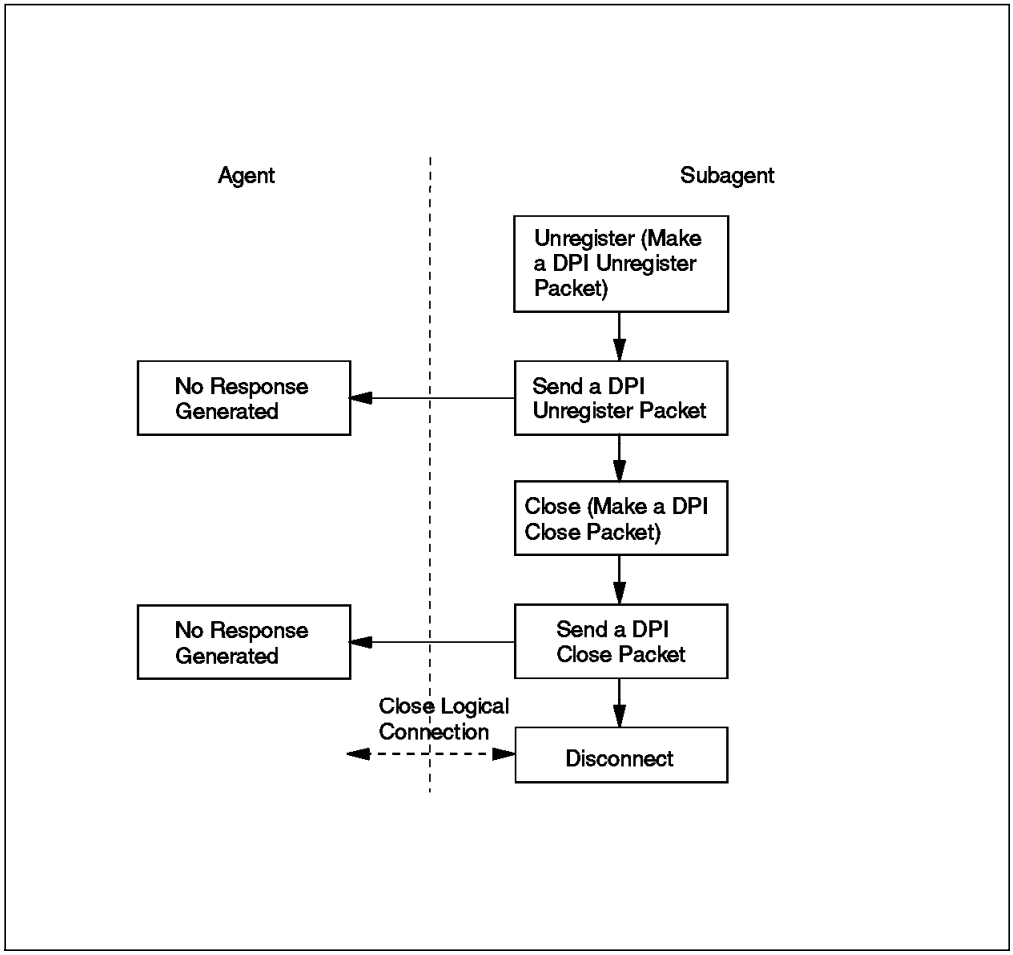


Figure 80. SNMP Subagent Unregister and Close Connection

Figure 80 shows the SNMP subagent unregistering itself from the SNMP agent and then closing the connection.

Note: All DPI subagent functions are performed within the logical bracket formed by a CONNECT and DISCONNECT.

At any given time, a subagent may have:

- 0 or 1 connections with the SNMP agent
- 0 or 1 opens with the SNMP agent
- 0 or n registrations

Zero opens is not a useful normal condition for a subagent and would occur only briefly between a DPI CONNECT request and the sending of a DPI OPEN packet.

Zero registrations will only be useful for a subagent that only sends traps and does not implement any MIB groups.

More than a single subtree registration would be useful for a variety of reasons:

1. Perhaps separate MIBs are being implemented by separate people, but it was decided to have these MIB implementations all run within a single job.
2. Perhaps a few OIDs within a MIB are known to have a relatively large overhead, compared to the rest of the subtree, for DPI request type. There

may be performance benefits to registering these OIDs separately (from the rest of the subtree) with a different timeout value.

7.4 Description of SNMP Subagent DPI Requests

In this section we give a detailed description of each of the OS/400 SNMP subagent DPI requests. The requests are described in the same order as shown in 7.3, “DPI Agent Requests” on page 92. The flowcharts in 7.3, “DPI Agent Requests” on page 92 provide an example of how these requests can be used.

For each DPI request, we will mention which API performs the request, if there is one available. For a more detailed explanation of each API, refer to the Simple Network Management Protocol (SNMP) Subagent APIs section of the *OS/400 System API Reference: UNIX Type APIs* book (SC41-3875 for V3R2 or SC41-4875 for V3R6).

7.4.1 Subagent Programming Concepts

The DPI API for OS/400 is the 2.0 level of the protocol. This is designed to be highly compatible with SNMPv2 (see Appendix F, “SNMPv2” on page 297 for more information) even though the SNMP agent is SNMPv1. This compatibility facilitates future upgrades of a subagent implementation for use with an SNMPv2 agent.

7.4.2 SNMP Agent Initiated DPI Requests

The following are the DPI requests that are initiated by an SNMP agent:

- **GET Processing**

  The SNMP agent sends a DPI GET packet specifying a request for one or more MIB variables that the subagent has taken responsibility for.

  If the subagent encounters an error while processing the request, it creates a DPI RESPONSE packet with an appropriate error indication in the error_code field.

  If there are no errors, the subagent creates a DPI RESPONSE packet including the OID, type, length and value of the MIB variable requested.

  The DPI RESPONSE packet is sent to the agent.

  The following chart shows the flow of a GET request originated by the SNMP agent.

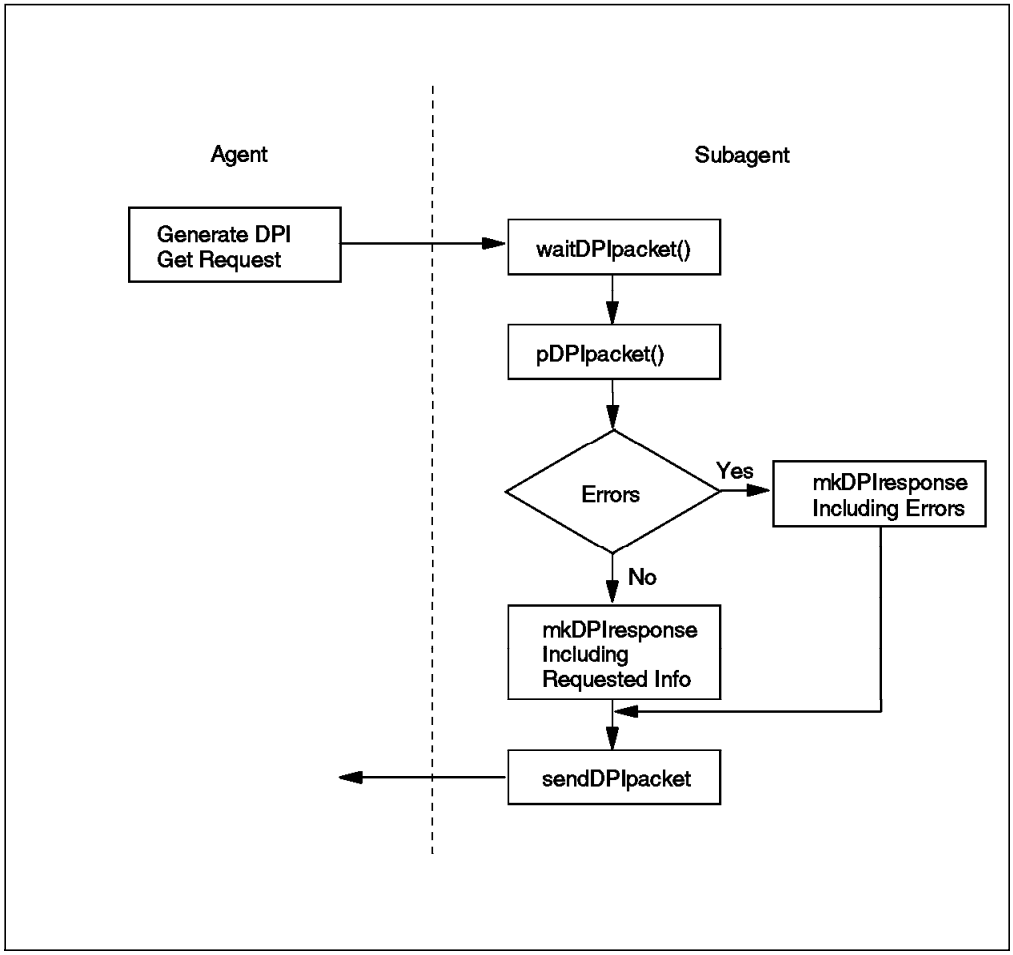


Figure 81. SNMP Subagent API GET Processing

The **waitDPIpacket()** API corresponds to the wait operation. The subagent must be in waiting status to receive requests or responses from the agent. See 7.4.4, “SNMP Subagent Communication Functions with SNMP Agent” on page 104 for more information.

The **pDPIpacket()** API parses the DPI packet received to make it available for processing by the subagent.

The **mkDPIresponse()** API is the one that constructs the DPI response packet. See 7.4.3, “SNMP Subagent Initiated DPI Requests” on page 103 for more information.

The **sendDPIpacket()** API sends the response back to the agent. See 7.4.4, “SNMP Subagent Communication Functions with SNMP Agent” on page 104 for more information.

- **GETNEXT Processing**

  The SNMP agent sends a DPI GETNEXT packet specifying the objects on which the GETNEXT operation is going to be performed. For this operation the subagent is to return the OID, type, length and value of the following MIB variable to the specified variable.

- **SET Processing**

  The SNMP agent sends a DPI SET packet specifying a request for one or more MIB variables to be set, and the value to be set.

If the subagent encounters an error while processing the request, it creates a DPI RESPONSE packet with an appropriate error indication in the error_code field.

If there are no errors, the subagent creates a DPI RESPONSE packet in which the error_code is set to SNMP_ERROR_noError (zero) and the error_index is set to zero.

The DPI SET will not be executed until the agent sends the DPI COMMIT or DPI UNDO packet. When the subagent encounters no errors in the DPI SET packet it sends the DPI RESPONSE packet and allocates the required resources and prepares itself for the SET.

The agent analyzes the DPI RESPONSE packet and if there are no errors it sends the DPI COMMIT to the subagent. The DPI COMMIT will have the same information as specified in the DPI SET request. The subagent can now carry out the SET phase.

If the subagent encounters no error while performing the DPI COMMIT, the subagent creates a DPI RESPONSE packet in which the error_code is set to SNMP_ERROR_noError (zero) and the error_index is set to zero.

If the agent encounters an error in the DPI RESPONSE to the SET request or to the COMMIT request, it will issue the UNDO request.

The agent request may involve changing the content of different MIB variables; these variables may or may not belong to the same subagent. The agent must verify that all the DPI RESPONSEs from the same or different subagents have no errors. If any one of them has an error, the agent should send an UNDO request to all the subagents involved in the request.

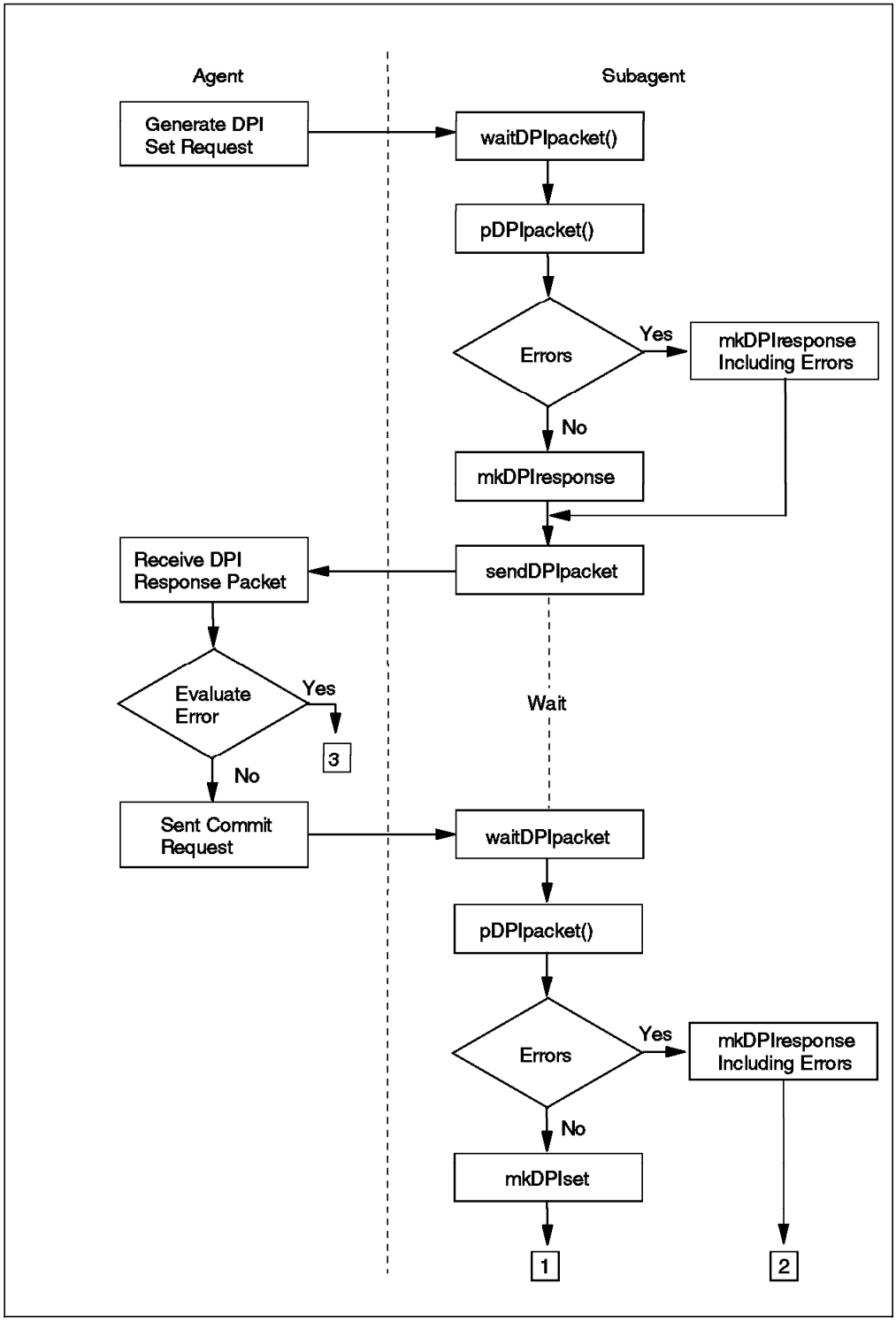


Figure 82. SNMP Subagent API SET Processing (Part 1)

Figure 82 and Figure 83 on page 101 show the sequence of events involved is DPI set processing between the subagent and agent.

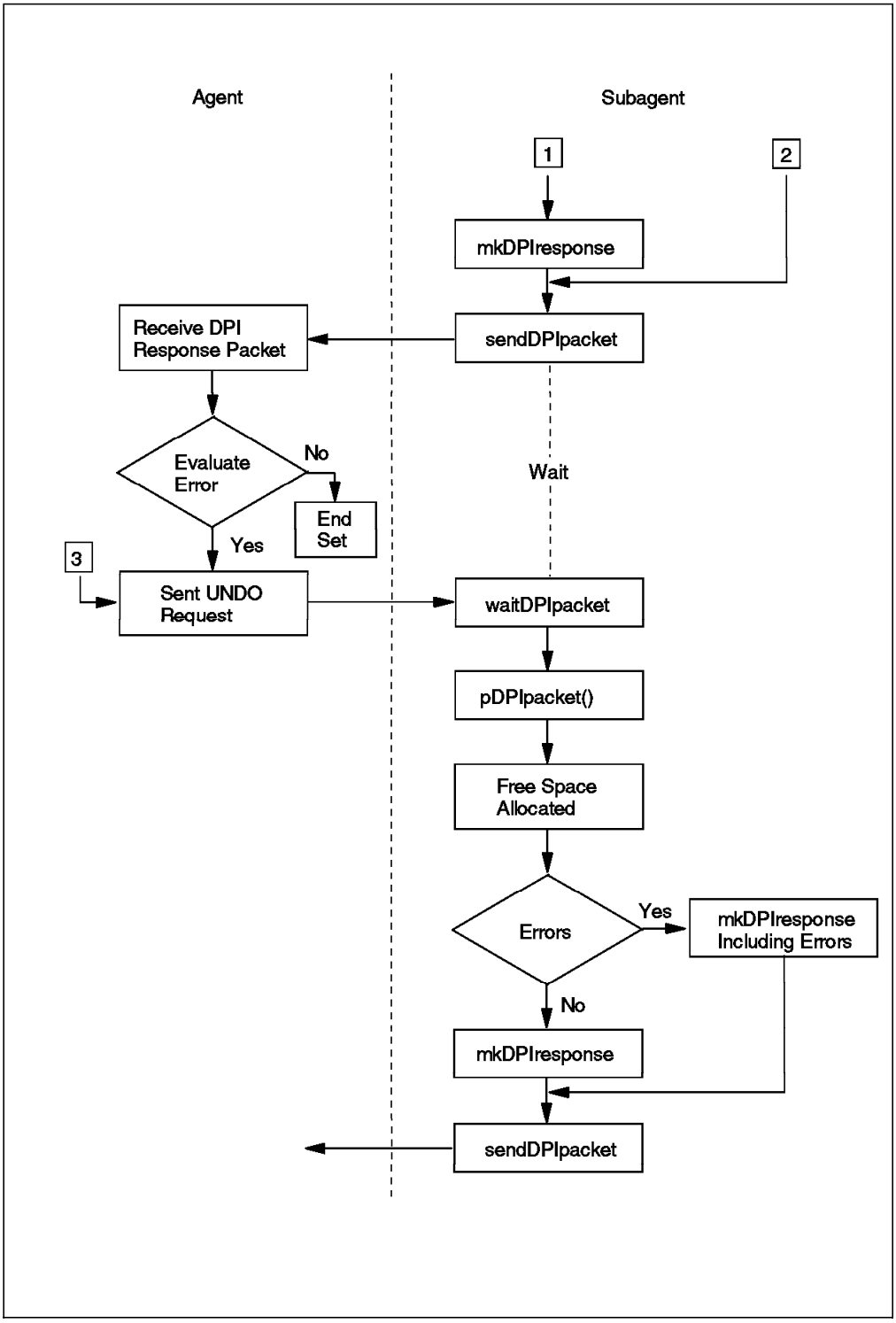


Figure 83. SNMP Subagent API SET Processing (Part 2)

The **waitDPIpacket()** API corresponds to the wait operation. The subagent must be in the waiting status to receive requests or responses from the agent. See 7.4.4, “SNMP Subagent Communication Functions with SNMP Agent” on page 104 for more information.

The **pDPIpacket()** API parses the DPI packet received to make it available for processing by the subagent.

The **mkDPIresponse()** API is the one that constructs the DPI response packet. See 7.4.3, “SNMP Subagent Initiated DPI Requests” on page 103 for more information.

The **sendDPIpacket()** API sends the response back to the agent. See 7.4.4, “SNMP Subagent Communication Functions with SNMP Agent” on page 104 for more information.

The **mkDPIset()** API makes a DPI set structure and adds it to a chained list of set structures if previous calls have been made.

For more information on DPI COMMIT and DPI UNDO, refer to the following topics.

- **COMMIT**

  The DPI COMMIT is requested by the agent when it receives a no error DPI RESPONSE for a DPI SET from that subagent and all the other subagents specified in the original request. The DPI COMMIT tells the subagent that the changes requested in the original request can be carried out.

  If the subagent encounters an error while processing the COMMIT request, it creates a DPI RESPONSE packet indicating the error condition.

  If there are no errors, the subagent creates a DPI RESPONSE.

- **UNDO**

  The DPI UNDO is issued by the agent when it receives an error condition in the DPI RESPONSE packet from that subagent or from any other subagent involved in the original request.

  If the subagent encounters an error while performing the UNDO request, the subagent creates a DPI RESPONSE indicating the error condition.

  If there are no errors, the subagent creates a DPI RESPONSE.

- **UNREGISTER**

  A subagent may unregister a previously registered subtree. Going through the DPI REGISTER request explained in 7.4.3, “SNMP Subagent Initiated DPI Requests” on page 103 will help you understand the DPI UNREGISTER request.

  A subagent may receive a DPI UNREGISTER packet from the agent. The following are some reasons this may happen:

  The agent is terminating.

  A prior DPI packet that the agent sent to the subagent did not get a response in time, so the agent is unregistering the subagent.

  If an SNMP manager sets a subagent MIB OID for this subagent′s subtree to invalid (refer to 9.3, “SNMP Subagent MIB” on page 158 for more information on the subagent MIB) the agent will unregister the subagent.

  For more information on DPI UNREGISTER requests originated by the subagent, refer to 7.4.3, “SNMP Subagent Initiated DPI Requests” on page 103.

- **CLOSE**

  A subagent will receive a DPI CLOSE packet from the agent whenever the following occur:

  The DPI OPEN request receives an error.

If an SNMP manager sets an invalid OID in the Subagent MIB's saTable (refer to 9.3, "SNMP Subagent MIB" on page 158 for more information on subagent MIB) the agent will send DPI CLOSE packet to the subagent.

A timeout is encountered.

The packet will contain a reason code for the CLOSE request. A subagent does not have to send a response to the CLOSE request. The agent just assumes that the subagent will handle it appropriately. The close takes place regardless of what the subagent does with it.

The API that generates the DPI CLOSE packet is the **mkDPIclose()** API.

For more information on DPI CLOSE requests originated by the subagent refer to 7.4.3, "SNMP Subagent Initiated DPI Requests."

7.4.3 SNMP Subagent Initiated DPI Requests

The following are the DPI requests initiated by an SNMP subagent:

- **OPEN**

  After a successful connection is made, a subagent must open a connection with the agent. To do so, it must send a DPI OPEN packet.

  Once a subagent has sent a DPI OPEN packet to an agent, it should expect a DPI RESPONSE packet that informs the subagent about the result of the request. The packet ID of the RESPONSE packet should be the same as that of the OPEN packet to which the RESPONSE packet is the reply.

  If an error RESPONSE is received on the OPEN packet, a DPI CLOSE packet with an error will also be received. In this situation, the agent closes the connection.

  If the OPEN is accepted, the next step is to REGISTER one or more MIB subtrees.

  The **mkDPIopen()** API is the one that generates the DPI OPEN packet that is then sent to the agent.

- **CLOSE**

  When a subagent is finished and wants to end processing, it should first UNREGISTER its subtrees and then close the connection with the agent. To do so it must send a DPI CLOSE packet, which specifies a reason for closing. A response to the CLOSE request should not be expected.

  The **mkDPIclose()** API is the one that generates the DPI CLOSE packet that is then sent to the agent. As a result of sending the packet, the DPI connection will be closed.

  For more information on DPI CLOSE request originated by the agent refer to 7.4.2, "SNMP Agent Initiated DPI Requests" on page 97.

- **REGISTER**

  Before a subagent can receive any requests for MIB variables, it must first register the variables or subtree it supports with the SNMP agent.

  Once a subagent has sent a DPI REGISTER packet to the agent, it should expect a DPI RESPONSE packet that informs the subagent about the result of the request. The packet ID of the RESPONSE packet should be the same as that of the REGISTER packet to which the RESPONSE packet is the reply.

The **mkDPIregister()** API is the one that generates the DPI REGISTER packet that is then sent to the agent.

- **UNREGISTER**

  A subagent may unregistered a previously registered subtree.

  Some of the reasons a subagent may want to unregister a previously registered subtree are:

  If the subagent is a proxy agent for yet another program which is no longer running, to prevent unnecessary traffic the subagent unregisters it.

  To change a registration parameter such as timeout, priority. (the subagent will unregister and then register again with new values)

  Although the **mkDPIclose()** API implicitly unregisters the subtree, the DPI UNREGISTER packet can be sent before a DPI CLOSE packet.

  Once a subagent has sent a DPI UNREGISTER packet to the agent, it should expect a DPI RESPONSE packet that informs the subagent about the result of the request. The packet ID of the RESPONSE packet should be the same as that of the UNREGISTER packet to which the RESPONSE packet is the reply.

- **TRAP**

  A subagent can request that the SNMP agent generates a trap for it. The subagent must provide the desired values for the generic and specific parameters of the trap. It may optionally provide one or more OID, type, length, or value parameters that will be included in the trap packet.

  It may optionally specify an Enterprise ID for the trap to be generated.

  The **mkDPItrap()** API generates the DPI TRAP packet that is then sent to the agent.

7.4.4 SNMP Subagent Communication Functions with SNMP Agent

The following are the DPI requests that are initiated by an SNMP subagent to establish/end communications with an SNMP agent:

- **CONNECT**

  The DPI CONNECT request is the very first subagent request that the subagent has to issue to establish communications. It establishes a logical connection between the SNMP subagent and the local (running on the same AS/400 as the subagent) SNMP agent and prepares for the next logical subagent event, which is to perform a DPI OPEN function.

  The **connectSNMP()** API is the one that establishes the logical connection between the SNMP subagent and the SNMP agent (in the same AS/400). One of the parameters of the **connectSNMP()** API specifies a data queue that the subagent wants the SNMP agent to use when sending work to the subagent. This data queue name is used by the agent as part of the identify of a subagent. Hence, only one subagent may use a particular data queue at any one time and a subagent can have a single data queue. This API will not create the data queue for the subagent, it should already be created when this call is made.

- **DISCONNECT**

The DPI DISCONNECT request is the logical opposite of the connect function and is the very last subagent API that is used. It ends or closes the logical connection between the SNMP agent and subagent.

The **disconnectSNMP()** API is the one that ends the logical connection between the subagent and the SNMP agent.

- **WAIT**

  The main purpose for which an SNMP subagent is developed is to provide additional MIB groups to SNMP manager applications. Therefore, an SNMP subagent spends most of its time waiting for (and processing) requests that an SNMP manager has sent to the local SNMP agent.

  When a request arrives from the SNMP agent the **waitDPIpacket()** API will receive it, verify that it is from the SNMP agent, and if not return it to the caller. The DPI packet is then parsed using the **pDPIpacket()** API so that its content is available.

  The **waitDPIpacket()** API completely handles the subagent's data queue, logically doing these steps:

  1. Check data queue for incoming messages.
  2. When data queue has a message, receive it.
  3. Check the message; if it is not from the SNMP agent, return it to the caller.
  4. If the message is from the SNMP agent, copy the DPI packet to the subagent buffer.

- **RECEIVE**

  The DPI RECEIVE is an alternative way of getting a response and work from the SNMP agent.

  The **receiveDPIpacket()** API is used to receive responses and work from an SNMP agent. The difference between the DPI WAIT and the DPI RECEIVE is that the DPI RECEIVE packet only verifies if the message is from the SNMP agent and then copies the DPI packet to the buffer. In the case where a DPI RECEIVE is being used, the subagent application must continuously check the data queue. When there is a message on the queue, receive it, and check if the message is from the SNMP agent. If not, return it to caller. After all these steps have been done, the **receiveDPIpacket()** API can be used to process the DPI RECEIVE packet.

- **SEND**

  The DPI SEND packet is used to send any type of DPI packet the subagent wants to send to the SNMP agent.

  The **sendDPIpacket()** API is the one that sends a copy of the DPI packet to the SNMP agent (on the same AS/400 as the subagent).

7.5 SNMP Subagent APIs

The SNMP subagent APIs can be used to dynamically extend the Management Information Base (MIB) that the SNMP agent is aware of. The MIB is extended, without any change to the SNMP agent itself, while the AS/400 is running. Dynamically added MIB subtrees (as defined and supported by a program known as a subagent) provide this capability. The remote and automated system capabilities of the AS/400 within the framework may be extended. For example,

an SNMP MIB group for an RPG or SQL application may be defined, and then SNMP protocol data units (PDUs) may be used, such as get and set, to determine status information or to make changes in control variables.

The Distributed Protocol Interface (DPI) is an extension to the SNMP agent that permits users to dynamically add, delete or replace management variables in the local MIB without requiring recompilation of the SNMP agent.

7.5.1 Subagent API Functions

Most of the APIs were discussed in topic 7.4, "Description of SNMP Subagent DPI Requests" on page 97. In Table 2 all the SNMP subagent APIs are shown and a description of them is provided. For a more detailed explanation of the SNMP subagent API, refer to the *OS/400 System API Reference: UNIX Type APIs* book (SC41-3875 for V3R2 or SC41-4875 for V3R6).

Table 2. SNMP Subagent APIs

Operation	**Description**
connectSNMP()	Establish connection with SNMP agent
debugDPI()	Set DPI packet trace
disconnectSNMP()	End connection with SNMP agent
DPI_PACKET_LEN()	Get length of DPI packet
fDPIparse()	Free storage from DPI packet parse
fDPIset()	Free storage from DPI set packet
mkDPIAreYouThere()	Make a DPI AreYouThere packet
mkDPIclose()	Make a DPI close packet
mkDPIopen()	Make a DPI open packet
mkDPIregister()	Make a DPI register packet
mkDPIresponse()	Make a DPI response packet
mkDPIset()	Make a DPI set packet
mkDPItrap()	Make a DPI trap packet
mkDPIunregister()	Make a DPI unregister packet
pDPIpacket()	Parse a DPI packet
receiveDPIpacket()	Receive a DPI packet from the SNMP agent
sendDPIpacket()	Send a DPI packet to the SNMP agent
waitDPIpacket()	Wait for a DPI packet

Note:

- These functions use header (include) files from the library QSYSINC, which is optionally installable. QSYSINC is provided with OS/400 (5763-SS1) option 13 (Openness Includes). Make sure QSYSINC is installed on your system before using any of the functions. All of the SNMP subagent APIs use header file qtossapi.h. You can see this in source file H, member QTOSSAPI, in library QSYSINC.
- Programs using the SNMP subagent API must be written in the C language. ILE C/400 can be used to compile the program with the Create C Module (CRTCMOD) CL command. After the *MOD object is created, specify DNDSVRPGM(QTOSSAPI) in the Create Program (CRTPGM) CL command. For

additional information refer to *ILE C/400 Programmer's Guide, SC09-1820*, and the *ILE C/400 Programmer's Reference*, SC09-1281.

7.6 SNMP Subagent Example

The following program is an example of an SNMP subagent. In this example it can be seen:

1. How the subagent connects and registers with the agent.
2. How the agent executes a GET, GETNEXT and SET operation on the subagent.
3. How the connection is ended between the agent and subagent.

```
#include <pointer.h>             /* pointer types                    */
#include <signal.h>              /* signal and _GetExcData functions */
#include <except.h>              /* definition used by _GetExcData   */
#include <stdio.h>               /* printf function                  */
#include <stdlib.h>              /* random or srand or randomize, abs*/
#include <time.h>
#include <string.h>              /* can't rock & roll without this 1 */
#include <ctype.h>
#include  "qtossapi.h"           /* the subagent include         */



/* ------------prototypes------------------------------
*/
static int     init_subagent(void),
               do_open(void),
               do_register(int, char *, int),
               do_trap(void),
               handle_get(snmp_dpi_hdr *,snmp_dpi_get_packet *),
               handle_set(snmp_dpi_hdr *,snmp_dpi_set_packet *),
               handle_commit(snmp_dpi_hdr *,snmp_dpi_set_packet *),
               handle_undo(snmp_dpi_hdr *,snmp_dpi_set_packet *),
               handle_next(snmp_dpi_hdr *,snmp_dpi_next_packet *),
               handle_unreg(snmp_dpi_hdr *,snmp_dpi_ureg_packet *),
               handle_close(snmp_dpi_hdr *,snmp_dpi_close_packet *),
               handle_resp(snmp_dpi_hdr *,snmp_dpi_resp_packet *),
               handle_expected_response( int sent_packet_type ),
               do_ayt(void),
               register_sa(void);



/*********************************************************************/
/* This SAsample's OBJECT IDENTIFIERS                                */
/*********************************************************************/
#define DPI_SIMPLE_INTEGER    "1.0"
#define DPI_SIMPLE_STRING     "2.0"
#define DPI_SIMPLE_OID        "3.0"
#define DPI_SIMPLE_NULL       "4.0"
#define DPI_SIMPLE_IPADDRESS  "5.0"
#define DPI_SIMPLE_COUNTER    "6.0"
#define DPI_SIMPLE_GAUGE      "7.0"       /* dpiSimpleGuage.0        */
#define DPI_SIMPLE_TIMETICKS  "8.0"       /* dpiSimpleTimeticks.0    */
#define DPI_SIMPLE_OPAQUE     "9.0"       /* dpiSimpleOpaque.0       */



/* Integer32  (1.0) */
#define              value1        cur_val1
static long int      cur_val1    = 1;
static long int      new_val1    = 0;
static long int      old_val1    = 0;

/* Display String  (2.0) */
#define              value2_p      cur_val2_p
static char         *cur_val2_p   = "01234567";
static char         *new_val2_p   = (char *)0;
static char         *old_val2_p   = (char *)0;

/* Oid  (3.0) */
#define              value3_p      cur_val3_p
```

```
static char         *cur_val3_p   = "1.3.6.29.28.27.26";
static char         *new_val3_p   = (char *)0;
static char         *old_val3_p   = (char *)0;

/* Null  (4.0) */
#define              value4_p      cur_val4_p
static char         *cur_val4_p   = (char *)0;
static char         *new_val4_p   = (char *)0;
static char         *old_val4_p   = (char *)0;

/* Ipaddress (5.0) */
#define              value5        cur_val5
static unsigned long cur_val5    = 5;
static unsigned long new_val5    = 0;
static unsigned long old_val5    = 0;

/* Counter (6.0) */
#define              value6        cur_val6
static unsigned long cur_val6    = 6;
static unsigned long new_val6    = 0;
static unsigned long old_val6    = 0;

/* Gauge (7.0) */
#define              value7        cur_val7
static unsigned long cur_val7    = 7;
static unsigned long new_val7    = 0;
static unsigned long old_val7    = 0;

/* Timeticks (8.0) */
#define              value8        cur_val8
static unsigned long cur_val8    = 8;
static unsigned long new_val8    = 0;
static unsigned long old_val8    = 0;

/* Opaque (9.0) */
#define              value9_p      cur_val9_p
static char         *cur_val9_p   = ")*(@\OO#$H\OVOI\n{R%}wO4tO";
static char         *new_val9_p   = (char *)0;
static char         *old_val9_p   = (char *)0;




/* msa subagent stuff */
#define        max_exceptions      30
#define        this_sa_timeout     1800     /* in seconds, 1800=30min used
                                               for QTOSwaitDPIpacket call
                                               (not the same as register
                                               timeout! (or open timeout!!))
                                            */
#define        this_sa_dataq_nam   "*MSA_Q     "
#define        This_sa_libname     "SATEST"
#define        This_sa_qname       "SATESTQ"
#define        this_sa_domain      snmpNMQdomain
#define        packetidok          30
#define        packetidNok         107
#define        gotitmsg            "msa->ma msg <....5....0> "



typedef _Packed struct {
        short int       len;
        char            proto_major;
        char            proto_minor;
        char            proto_release;
        short int       packet_id;
        char            packet_type;
        char            data[1];
        } DPI_WireFmt;

enum { Off, On };

static int     rc,
               exception_counter                    = 0,
               num_traps_sent                       = 0,
               resp_msg_index                       = -1,
               this_sa_register_timeout             = 18,
               this_sa_max_varbinds                 = 1,
               special_oid_switch                   = Off,
               dpitracelevel                        = 1
               ;
```

```
#define      DPI_SIMPLE_MIB   "1.3.6.1.2.3.4.5.6."
#define      Defaultoid       "1.3.6.1.2.3.4.5.6"
#define      maxoidlen        256
#define      max_msg_len      2048


static char
               msg_area[max_msg_len],

              *this_sa_oid        = Defaultoid,
              *this_sa_subtree    = DPI_SIMPLE_MIB,
              *this_sa_desc       = "SAsample DPI subagent description",
              *exp_r_buf_p        = msg_area;


static unsigned long int    rcvd_msglen;
static unsigned char       *openpacket;




/*-------------------------------------------------------------------
|
|   Name:  main for a SAsample DPI subagent
|
|   (exception handling has been omitted for simplicity)
|
 -------------------------------------------------------------------*/
void main( int argc, char *argv[]) {

   int              i, j, rc,
                    trap_flag,
                    num_msgs_in = 0;
   long int         respcode;
   char            *cp1;
   snmp_dpi_hdr    *dpiheader,
                   *hdr_p;



   /* Handle parameters, as needed. Some example parms that might
      be useful are;

         subagent description, subagent's own OID, timeout,
         max varbinds per DPI packet, DPI trace level,
         what subtree(s) to register, etc.
   */


   init_subagent();
   debugDPI(dpitracelevel);      /* turn on DPI debugging */



   /*-------------------------------------------------------------
      first, connect to the local SNMP agent
   */
   rc = connectSNMP( This_sa_qname, This_sa_libname, 35 );
   if (rc !=  snmpsa_RC_ok )  {
       /* handle exceptions */
       return;
       }



   /*-------------------------------------------------------------
      do DPI open
   */
   rc = do_open();
   if ( rc != snmpsa_RC_ok ) { /* handle exception */  return; }



   /*-------------------------------------------------------------
       do DPI register of subtree's
   */
   if ( (rc = do_register( this_sa_register_timeout,  this_sa_subtree, 0 ))
         != 0) return;
```

```
/*-------------------------------------------------------------
   subagent's working loop
*/
for (;;) {


    if ((rc=waitDPIpacket( this_sa_timeout,
                    msg_area,
                    &rcvd_msglen     )) == SNMP_ERROR_DPI_noError ) {


        /*-----------------------------------------------------------
           got a DPI packet ok ... lets process it based on type
           of DPI packet
        */
        switch(hdr_p->packet_type) {     /* handle by DPI type      */
            case SNMP_DPI_GET:
                 rc = handle_get(hdr_p, hdr_p->data_u.get_p);
                 break;
            case SNMP_DPI_GETNEXT:
                 rc = handle_next(hdr_p,  hdr_p->data_u.next_p);
                 break;
            case SNMP_DPI_SET:
                 rc = handle_set(hdr_p,  hdr_p->data_u.set_p);
                 break;
            case SNMP_DPI_COMMIT:
                 rc = handle_commit(hdr_p, hdr_p->data_u.set_p);
                 break;
            case SNMP_DPI_UNDO:
                 rc = handle_undo(hdr_p, hdr_p->data_u.set_p);
                 break;
            case SNMP_DPI_CLOSE:
                 rc = handle_close(hdr_p, hdr_p->data_u.close_p);
                 break;
            case SNMP_DPI_UNREGISTER:
                 rc = handle_unreg(hdr_p, hdr_p->data_u.ureg_p);
                 break;
            case SNMP_DPI_RESPONSE:
                 rc = handle_resp(hdr_p, hdr_p->data_u.resp_p);
                 break;
            default:
                 printf("Unexpected DPI packet type %d\n",
                        hdr_p->packet_type);
                 rc = -1;
            } /* endswitch */

     if (rc!=0) {
            /* decide what to do -- maybe should end this
               subagent
            */
            printf("subagent function %d returned %d\n",
                   hdr_p->packet_type, rc );
            return;
         }
       } /* waitDPIpacket() was ok*/



    /*-------------------------------------------------------------
       here if some problem occurred with waitDPIpacket()
    */
    else {
        /* handle exception */
        }



    } /* for(ever loop)  */


/* SAsample ends -- close processing  */

} /* SAsample main() */
```

```
/*                    ----------------------
   --------------------                      ---------------------------
   --------------------    other routines    ---------------------------
   --------------------                      ---------------------------
                      ----------------------
*/



/*--------------------------------------------------------------------
*/
int init_subagent() {
   int  rc;
   char   *cmd =
 "CRTDTAQ DTAQ(" This_sa_libname "/" This_sa_qname ") MAXLEN(80) SEQ(*FIFO)";

   /* select lib and q names, create q, etc. */
   rc = system(cmd);
   return rc ;
   }



/*--------------------------------------------------------------------
*/
void delete_DTAQ() {
   system("DLTDTAQ DTAQ(SATEST/SATEST_Q)");
   return;
   }



/*--------------------------------------------------------------------
     do the open
*/
int do_open(void) {
   snmp_dpi_hdr  *hdr_p;

   openpacket = mkDPIopen(               /* Make DPI-OPEN packet    */
                    this_sa_oid,         /* Our identification      */
                    this_sa_desc,        /* Our description         */
                    5L,                  /* Our overall timeout     */
                    this_sa_max_varbinds, /* max varBinds/packet    */
                    DPI_NATIVE_CSET,     /* 13i new parm: sa c.set  */
                    0L,                  /* password length         */
                    (unsigned char *)0);  /* ptr to password        */


   if (openpacket == NULL) { return 1; }

   /* send open */
   if ((rc = sendDPIpacket(openpacket,DPI_PACKET_LEN(openpacket)))
                        !=snmpsa_RC_ok) {
       return 2;
       }

   return handle_expected_response( SNMP_DPI_OPEN );
   } /* do_open() */




/*--------------------------------------------------------------------
*/
static int do_register(int timeout, char *subtree_root, int garbflag ) {
   unsigned char *packet_p;
   int            rc, len, i;
   unsigned long  length;
   snmp_dpi_hdr  *hdr_p;
   char          *c_p;


   packet_p = mkDPIregister(             /* Make DPIregister packet */
                    timeout,             /* timeout in seconds      */
                    0,                   /* requested priority      */
                    subtree_root,        /* trailing '.'?           */
                    DPI_BULK_NO   );

   if (!packet_p) { return 1; }
```

```
   /* send it */
   if ((rc = sendDPIpacket(packet_p,DPI_PACKET_LEN(packet_p)))
                         !=snmpsa_RC_ok) {
           return 2;
           }

   return handle_expected_response( SNMP_DPI_REGISTER );
   } /* end of do_register() */



/*--------------------------------------------------------------------
     send an Are_You_There packet
*/
static int do_ayt(void) {
   unsigned char *packet_p;
   int            rc;
   unsigned long  length;
   snmp_dpi_hdr  *hdr_p;

   packet_p = mkDPIayt();

   if (!packet_p) return -1;

   if ((rc=sendDPIpacket(packet_p,DPI_PACKET_LEN(packet_p)))
                          !=snmpsa_RC_ok) {
       /* msg */
       return rc;
       }

   return handle_expected_response( SNMP_DPI_ARE_YOU_THERE );
   } /* do_ayt() */



/*-------------------------------------------------------------
    handle a response DPI packet that is expected back due to
    some subagent-initiated DPI packet
*/
int handle_expected_response( int sent_packet_type ) {
   int              rc, len, i;
   unsigned long    length;
   snmp_dpi_hdr    *hdr_p;


   /* see what the SNMP agent has to say back (if anything) */
   rc = waitDPIpacket( this_sa_timeout, exp_r_buf_p, &length );

   if (rc == snmpsa_RC_timedout ) {
       /* msg */
       return -2;
       }

   if (rc != SNMP_ERROR_DPI_noError) {
       /* msg */
       return -3;
       }

   hdr_p = pDPIpacket((unsigned char*)exp_r_buf_p);
                                          /* parse DPI packet */
   if (hdr_p == snmp_dpi_hdr_NULL_p) {
       /* msg */
       return -4;
       }

   if (hdr_p->packet_type != SNMP_DPI_RESPONSE) {
       /* msg */
       return -5;
       }

   rc = hdr_p->data_u.resp_p->error_code;
   if (rc != SNMP_ERROR_DPI_noError) {
       return -6;
       }

   return snmpsa_RC_ok;
   }
```

```
/*--------------------------------------------------------------------
|
|
|     following routines adapted from dpi_sample.c
|
|
*/

/*********************************************************************/
/* Processing a GET request                                          */
/*********************************************************************/
/*                                                                   */
/* A get request is pretty easy to process. When the DPI packet is   */
/* parsed, the snmp_dpi_hdr structure will show in the packet_type   */
/* that this is a SNMP_DPI_GET packet. In that case, the data_u field*/
/* contains a ptr to a GET-varBind, which is represented in an       */
/* snmp_dpi_get_packet structure:                                    */
/*                                                                   */
/*   struct dpi_get_packet {                                         */
/*     char                  *object_p;    / * ptr to OIDstring     */
/*     char                  *group_p;     / * ptr to subtree       */
/*     char                  *instance_p;  / * ptr to rest of OID   */
/*     struct dpi_get_packet  *next_p;     / * ptr to next in chain */
/*   };                                                              */
/*   typedef struct dpi_get_packet       snmp_dpi_get_packet;        */
/*   #define snmp_dpi_get_packet_NULL_p  ((snmp_dpi_get_packet *)0)  */
/*                                                                   */
/* So, assuming we have registered example subtree dpiSimpleMIB      */
/* and a GET request comes in for one variable dpiSimpleInteger.0    */
/* (so that is object 1 instance 0 in our subtree), then the fields  */
/* in the snmp_dpi_get_packet would have ptrs that point to:         */
/*                                                                   */
/*   object_p    ->  "1.3.6.1.4.1.2.2.1.5.1.0"                       */
/*   group_p     ->  "1.3.6.1.4.1.2.2.1.5"                           */
/*   instance_p  ->  "1.0"                                           */
/*   next_p      ->  snmp_dpi_get_packet_NULL_p                      */
/*                                                                   */
/* If there are multiple varBinds in a GET request, then each one    */
/* is represented in a snmp_dpi_get_packet structure and all the     */
/* snmp_dpi_get_packet structures are chained via the next ptr.      */
/* As long as the next ptr is not the snmp_dpi_get_packet_NULL_p     */
/* pointer then there are more varBinds in the list.                 */
/*                                                                   */
/* Now we can analyze the varBind structure for whatever checking we */
/* want to do. Once we are ready to make a response that contains    */
/* the value of the variable, then we first prepare a SET-varBind    */
/* which is represented in an snmp_dpi_set_packet structure:         */
/*                                                                   */
/*   struct dpi_set_packet {                                         */
/*     char                  *object_p;    / * ptr to OIDstring     */
/*     char                  *group_p;     / * ptr to subtree       */
/*     char                  *instance_p;  / * ptr to rest of OID   */
/*     unsigned char          value_type;  / * SNMP_TYPE_xxxx       */
/*     unsigned short         value_len;   / * value length         */
/*     char                  *value_p;     / * ptr to value itself  */
/*     struct dpi_set_packet  *next_p;     / * ptr to next in chain */
/*   };                                                              */
/*   typedef struct dpi_set_packet       snmp_dpi_set_packet;        */
/*   #define snmp_dpi_set_packet_NULL_p  ((snmp_dpi_set_packet *)0)  */
/*                                                                   */
/* We can use the mkDPIset() function to prepare such a structure.   */
/* This function expects the following arguments:                    */
/*  - A ptr to an existing snmp_dpi_set_packet structure if the new  */
/*    varBind must be added to an existing chain of varBinds.        */
/*    If this is the first (or the only) varBind in the chain, then  */
/*    pass the snmp_dpi_set_packet_NULL_p ptr to indicate this.      */
/*  - a ptr to the subtree that we registered.                       */
/*  - a ptr to the rest of the OID, in other words the piece that    */
/*    follows the subtree.                                           */
/*  - the value type of the value to be bound to the variable name.  */
/*    This is must be one of the SNMP_TYPE_xxx values as defined in  */
/*    the snmp_dpi.h include file.                                   */
/*  - the length of the value (for integer type values this must be  */
/*    a length of 4. So we always work with 32-bit signed or         */
/*    unsigned integers (except of course for the Counter64 type,    */
/*    for those we must point to a snmp_dpi_u64 structure and pass   */
/*    the length of that structure).                                 */
/*  - a ptr to the value.                                            */
/* Memory for the varBind is dynamically allocated and the data      */
/* itself is copied. So upon return we can dispose of our own ptrs   */
/* and allocated memory as we please. If the call is successful,     */
/* then a ptr is returned:                                           */
```

```
/*  - to a new snmp_dpi_set_packet if it is the first/only varBind   */
/*  - to the existing snmp_dpi_set_packet that we passed on the call.*/
/*    In this case, the new packed has been chained to the end of    */
/*    the varBind list.                                              */
/* If the mkDPIset() call fails, a NULL ptr is returned.             */
/*                                                                   */
/* Once we have prepared the SET-varBind data, we can create a DPI   */
/* RESPONSE packet. To do so we can use the mkDPIresponse() function */
/* which expects these arguments:                                    */
/*  - a ptr to an snmp_dpi_hdr. We should use the hdr of the parsed  */
/*    incoming packet. It is used to copy the packet_id from the     */
/*    request into the response, such that the agent can correlate   */
/*    the response to a request.                                     */
/*  - a return code (snmp error code). If success, this should be    */
/*    SNMP_ERROR_noError (value zero). If failure, it must be one    */
/*    of the SNMP_ERROR_xxxx values as defined in the snmp_dpi.h     */
/*    include file.                                                  */
/*    Note that a request for a non-existing object or instance is   */
/*    not considered an error. Instead, we must pass a value type    */
/*    of SNMP_TYPE_noSuchObject or SNMP_TYPE_noSuchInstance          */
/*    respectively.                                                  */
/*    These 2 value types have an implicit value of NULL, so we can  */
/*    pass a zero length and a NULL ptr for the value in this case.  */
/*  - The index of the varBind in error (starts counting at 1).      */
/*    Pass zero if no error occured, else pass the proper index of   */
/*    the first varBind for which an error was detected.             */
/*  - a ptr to a (chain of) snmp_dpi_set_packets (varBinds) to be    */
/*    returned as response to the GET request.                       */
/*    If an error was detected, then an snmp_dpi_set_packet_NULL_p   */
/*    ptr may be passed.                                             */
/*                                                                   */
/* Here follows a code sample to return a response. We assume that   */
/* there are no errors in the request, but proper code should do the */
/* checking for that. For instance, we return a noSuchInstance if    */
/* the instance is not exactly what we expect and a noSuchObject if  */
/* the object instanceID is greater than 3 (like 4.0). But there     */
/* might be no instanceID at all, and we should check for that too.  */
/*********************************************************************/
static int handle_get( snmp_dpi_hdr *hdr_p,
                       snmp_dpi_get_packet *pack_p ) {

      unsigned char       *packet_p;
      int                  rc=0, value2_len, i, len,
                           something_extra = 0;
                           snmp_dpi_set_packet *varBind_p;
      char                *c_p;
      snmp_dpi_get_packet *packet_p_anchor;



   if (hdr_p == snmp_dpi_hdr_NULL_p ) {
          return(-1);
          }
   if (pack_p == snmp_dpi_get_packet_NULL_p ) {
          return(-2);
          }

   /* keep starting pointer */
   packet_p_anchor = pack_p; /* init with defn may fail due to iCC */

   varBind_p =                        /* init the varBind chain  */
          snmp_dpi_set_packet_NULL_p;  /* to a NULL pointer       */



   /*-----------------------------------------------------------------
   |   must handle multi-varbinds!!
   */
   do {

       if (pack_p->instance_p &&
           (strcmp(pack_p->instance_p,"1.0") == 0))
       {
         varBind_p = mkDPIset(            /* Make DPI set packet     */
                     varBind_p,           /* ptr to varBind chain    */
                     pack_p->group_p,     /* ptr to subtree          */
                     pack_p->instance_p,  /* ptr to rest of OID      */
                     SNMP_TYPE_Integer32, /* value type Integer 32   */
                     sizeof(value1),      /* length of value         */
                     &value1);            /* ptr to value            */

       } else if (pack_p->instance_p &&
           (strcmp(pack_p->instance_p,"2.0") == 0))
```

```
{


  varBind_p = mkDPIset(              /* Make DPI set packet  */
              varBind_p,             /* ptr to varBind chain */
              pack_p->group_p,       /* ptr to subtree       */
              pack_p->instance_p,    /* ptr to rest of OID   */
              SNMP_TYPE_DisplayString,/* value type          */
              strlen(value2_p),
              value2_p);             /* ptr to value         */

} else if (pack_p->instance_p &&
    (strcmp(pack_p->instance_p,"3.0") == 0))
{
  varBind_p = mkDPIset(              /* Make DPI set packet  */
              varBind_p,             /* ptr to varBind chain */
              pack_p->group_p,       /* ptr to subtree       */
              pack_p->instance_p,    /* ptr to rest of OID   */
              SNMP_TYPE_OBJECT_IDENTIFIER,/* value type      */
              strlen(value3_p),
              value3_p);


} else if (pack_p->instance_p &&
    (strcmp(pack_p->instance_p,"4.0") == 0))
{
  varBind_p = mkDPIset(              /* Make DPI set packet  */
              varBind_p,             /* ptr to varBind chain */
              pack_p->group_p,       /* ptr to subtree       */
              pack_p->instance_p,    /* ptr to rest of OID   */
              SNMP_TYPE_NULL,        /* value type           */
              strlen(value4_p),
              value4_p);             /* ptr to value         */

} else if (pack_p->instance_p &&
    (strcmp(pack_p->instance_p,"5.0") == 0))
{
  varBind_p = mkDPIset(              /* Make DPI set packet  */
              varBind_p,             /* ptr to varBind chain */
              pack_p->group_p,       /* ptr to subtree       */
              pack_p->instance_p,    /* ptr to rest of OID   */
              SNMP_TYPE_IpAddress,   /* value type           */
              sizeof(value5),        /* length of value      */
              &value5);              /* ptr to value         */

} else if (pack_p->instance_p &&
    (strcmp(pack_p->instance_p,"6.0") == 0))
{
  varBind_p = mkDPIset(              /* Make DPI set packet  */
              varBind_p,             /* ptr to varBind chain */
              pack_p->group_p,       /* ptr to subtree       */
              pack_p->instance_p,    /* ptr to rest of OID   */
              SNMP_TYPE_Counter32,   /* value type           */
              sizeof(value6),        /* length of value      */
              &value6);              /* ptr to value         */

} else if (pack_p->instance_p &&
    (strcmp(pack_p->instance_p,"7.0") == 0))
{
  varBind_p = mkDPIset(              /* Make DPI set packet  */
              varBind_p,             /* ptr to varBind chain */
              pack_p->group_p,       /* ptr to subtree       */
              pack_p->instance_p,    /* ptr to rest of OID   */
              SNMP_TYPE_Gauge32,     /* value type           */
              sizeof(value7),        /* length of value      */
              &value7);              /* ptr to value         */

} else if (pack_p->instance_p &&
    (strcmp(pack_p->instance_p,"8.0") == 0))
{
  varBind_p = mkDPIset(              /* Make DPI set packet  */
              varBind_p,             /* ptr to varBind chain */
              pack_p->group_p,       /* ptr to subtree       */
              pack_p->instance_p,    /* ptr to rest of OID   */
              SNMP_TYPE_TimeTicks,   /* value type           */
              sizeof(value8),        /* length of value      */
              &value8);              /* ptr to value         */

} else if (pack_p->instance_p &&
    (strcmp(pack_p->instance_p,"9.0") == 0))
{
  varBind_p = mkDPIset(              /* Make DPI set packet  */
              varBind_p,             /* ptr to varBind chain */
```

```
                    pack_p->group_p,        /* ptr to subtree       */
                    pack_p->instance_p,     /* ptr to rest of OID   */
                    SNMP_TYPE_Opaque,       /* value type           */
                    strlen(value9_p),
                    value9_p);              /* ptr to value         */


        } /* endif */


      varBind_p = mkDPIset( varBind_p,
                            pack_p->group_p,
                            pack_p->instance_p,
                            SNMP_TYPE_noSuchObject,
                            0L,
                            (unsigned char *)0        );


       pack_p = pack_p->next_p; /* to process any other varbinds */
       } while( pack_p != NULL );


   /*-------------------------------------------------------------
   */
   packet_p = mkDPIresponse(          /* Make DPIresponse packet */
                   hdr_p,                  /* ptr parsed request      */
                   SNMP_ERROR_noError,     /* all is OK, no error     */
                   0L,                     /* index is zero, no error */
                   varBind_p);             /* varBind response data   */

   if (!packet_p) return(-11);         /* If it failed, return    */



   /*-------------------------------------------------------------
       send it.
   */
   if ((rc=sendDPIpacket(packet_p, DPI_PACKET_LEN(packet_p)))
                                                 !=snmpsa_RC_ok) {
       printf("send of a response packet failed, rc=%d\n", rc);
       }

   return rc;
   } /* handle_get() */


/*********************************************************************/
/* Processing a GETNEXT request                                      */
/*********************************************************************/
/*                                                                   */
/* A getnext request is more difficult to process. When a DPI packet */
/* is parsed, the snmp_dpi_hdr structure shows in the packet_type    */
/* that this is a SNMP_DPI_GETNEXT packet, and so the data_u field   */
/* contains a ptr to a GETNEXT-varBind, which is represented in an   */
/* snmp_dpi_next_packet structure:                                   */
/*                                                                   */
/*   struct dpi_next_packet {                                        */
/*     char                  *object_p;    / * ptr to OIDstring     */
/*     char                  *group_p;     / * ptr to subtree       */
/*     char                  *instance_p;  / * ptr to rest of OID   */
/*     struct dpi_next_packet *next_p;     / * ptr to next in chain */
/*   };                                                              */
/*   typedef struct dpi_next_packet      snmp_dpi_next_packet;       */
/*   #define snmp_dpi_next_packet_NULL_p ((snmp_dpi_next_packet *)0) */
/*                                                                   */
/* So, assuming we have registered example subtree dpiSimpleMIB      */
/* and a GETNEXT arrives for one variable dpiSimpleInteger.0         */
/* (so that is object 1 instance 0 in our subtree), then the fields  */
/* in the snmp_dpi_get_packet structure would have ptrs pointing to: */
/*                                                                   */
/*   object_p    ->  "1.3.6.1.4.1.2.2.1.5.1.0"                       */
/*   group_p     ->  "1.3.6.1.4.1.2.2.1.5"                           */
/*   instance_p  ->  "1.0"                                           */
/*   next        ->  snmp_dpi_get_packet_NULL_p                      */
/*                                                                   */
/* If there are multiple varBinds in a GETNEXT request, then each    */
/* one is represented in a snmp_dpi_get_packet structure and all     */
/* the snmp_dpi_get_packet structures are chained via the next ptr.  */
```

```
/* As long as the next ptr is not the snmp_dpi_next_packet_NULL_p    */
/* pointer then there are more varBinds in the list.                 */
/*                                                                   */
/* Now we can analyze the varBind structure for whatever checking we */
/* want to do. Then we must find out which OID is the one that       */
/* lexicographically follows the one in the request. And it is that  */
/* OID with its value that we must return as a response. So we must  */
/* now also set the proper OID in the response.                      */
/* Once we are ready to make a response that contains the new OID    */
/* and the value of that variable, then we must first prepare a      */
/* SET-varBind which is represented in an snmp_dpi_set_packet:       */
/*                                                                   */
/*   struct dpi_set_packet {                                         */
/*     char                   *object_p;    / * ptr to OIDstring     */
/*     char                   *group_p;     / * ptr to subtree       */
/*     char                   *instance_p;  / * ptr to rest of OID   */
/*     unsigned char           value_type;  / * SNMP_TYPE_xxxx       */
/*     unsigned short          value_len;   / * value length         */
/*     char                   *value_p;     / * ptr to value itself  */
/*     struct dpi_set_packet  *next_p;      / * ptr to next in chain */
/*   };                                                              */
/*   typedef struct dpi_set_packet        snmp_dpi_set_packet;       */
/*   #define snmp_dpi_set_packet_NULL_p   ((snmp_dpi_set_packet *)0) */
/*                                                                   */
/* We can use the mkDPIset() function to prepare such a structure.   */
/* This function expects the following arguments:                    */
/*  - A ptr to an existing snmp_dpi_set_packet structure if the new  */
/*    varBind must be added to an existing chain of varBinds.        */
/*    If this is the first (or the only) varBind in the chain, then  */
/*    pass the snmp_dpi_set_packet_NULL_p ptr to indicate this.      */
/*  - a ptr to the subtree that we registered.                       */
/*  - a ptr to the rest of the OID, in other words the piece that    */
/*    follows the subtree.                                           */
/*  - the value type of the value to be bound to the variable name.  */
/*    This is must be one of the SNMP_TYPE_xxx values as defined in  */
/*    the snmp_dpi.h include file.                                   */
/*  - the length of the value (for integer type values this must be  */
/*    a length of 4. So we always work with 32-bit signed or         */
/*    unsigned integers (except of course for the Counter64 type,    */
/*    for those we must point to a snmp_dpi_u64 structure and pass   */
/*    the length of that structure).                                 */
/*  - a ptr to the value.                                            */
/* Memory for the varBind is dynamically allocated and the data      */
/* itself is copied. So upon return we can dispose of our own ptrs   */
/* and allocated memory as we please. If the call is successful,     */
/* then a ptr is returned:                                           */
/*  - to a new snmp_dpi_set_packet if it is the first/only varBind   */
/*  - to the existing snmp_dpi_set_packet that we passed on the call.*/
/*    In this case, the new packed has been chained to the end of    */
/*    the varBind list.                                              */
/* If the mkDPIset() call fails, a NULL ptr is returned.             */
/*                                                                   */
/* Once we have prepared the SET-varBind data, we can create a DPI   */
/* RESPONSE packet. To do so we can use the mkDPIresponse() function,*/
/* which expects these arguments:                                    */
/*  - a ptr to an snmp_dpi_hdr. We should use the hdr of the parsed  */
/*    incoming packet. It is used to copy the packet_id from the     */
/*    request into the response, such that the agent can correlate   */
/*    the response to a request.                                     */
/*  - a return code (snmp error code). If success, this should be    */
/*    SNMP_ERROR_noError (value zero). If failure, it must be one    */
/*    of the SNMP_ERROR_xxxx values as defined in the snmp_dpi.h     */
/*    include file.                                                  */
/*    Note that a request for a non-existing object or instance      */
/*    is not considered an error. Instead, we must pass the OID      */
/*    and value of the first OID that lexicographically follows      */
/*    the non-existing object and/or instance.                       */
/*    Also, reaching the end of our subtree (there is no NEXT OID)   */
/*    is not considered an error. In this situation we must return   */
/*    the original OID (as received in request) and a value_type     */
/*    of SNMP_TYPE_endOfMibView. This value_type has an implicit     */
/*    value of NULL, so we can pass a zero length and a NULL ptr     */
/*    for the value.                                                 */
/*  - The index of the first varBind in error (start counting at 1). */
/*    Pass zero if no error occurred, else pass the proper index of  */
/*    the first varBind for which an error was detected.             */
/*  - a ptr to a (chain of) snmp_dpi_set_packet(s) (varBinds) to     */
/*    be returned as response to the GETNEXT request.                */
/*    If an error was detected, then an snmp_dpi_set_packet_NULL_p   */
/*    ptr may be passed.                                             */
/*                                                                   */
/* Here follows a code sample to return a response. We assume that   */
/* there are no errors in the request, but proper code should do     */
```

```
/* the checking for that. We do proper checking for lexicographic    */
/* next object, but we do no checking for ULONG_MAX, or making sure  */
/* that the instance ID is indeed valid (digits and dots).           */
/* If we get to the end of our dpiSimpleMIB then we must return an   */
/* endOfMibView, as defined by the SNMPv2 rules.                     */
/*********************************************************************/
static int handle_next( snmp_dpi_hdr *hdr_p,
                        snmp_dpi_next_packet *pack_p)
{
       unsigned char       *packet_p;
       int                  rc=0,i;
       unsigned long        subid;        /* subid is unsigned       */
       unsigned long        instance;     /* same with instance      */
       char                *cp;
       snmp_dpi_set_packet *varBind_p;

       varBind_p =                        /* init the varBind chain  */
          snmp_dpi_set_packet_NULL_p;     /* to a NULL pointer       */



   /*-------------------------------------------------------------
   |   must handle multi-varbinds!!
   */
   do {


       if (pack_p->instance_p) {          /* we have an instance ID  */
          cp = pack_p->instance_p;        /* pick up ptr             */
          subid = strtoul(cp, &cp, 10);   /* convert subid (object)  */
          if (*cp == '.') {               /* followed by a dot ?     */
             cp++;                        /* point after it if yes   */
             instance=strtoul(cp,&cp,10); /* convert real instance   */
                                          /* not that we need it, we */
             subid++;                     /* only have instance 0,   */
                                          /* so NEXT is next object  */
             instance = 0;                /* and always instance 0   */
          } else {                        /* no real instance passed */
             instance = 0;                /* so we can use 0         */
             if (subid == 0) subid++;     /* if object 0, start at 1 */
          } /* endif */
       } else {                           /* no instance ID passed   */
          subid = 1;                      /* so do first object      */
          instance = 0;                   /* instance 0 (all we have)*/
       } /* endif */

       /* we have set subid and instance such that we can basically  */
       /* process the request as a GET now. Actually, we don't even  */
       /* need instance, because all out object instances are zero.  */


       switch (subid) {
       case 1:
         varBind_p = mkDPIset(            /* Make DPI set packet     */
                     varBind_p,           /* ptr to varBind chain    */
                     pack_p->group_p,     /* ptr to subtree          */
                     DPI_SIMPLE_INTEGER,  /* ptr to rest of OID      */
                     SNMP_TYPE_Integer32, /* value type Integer 32   */
                     sizeof(value1),      /* length of value         */
                     &value1);            /* ptr to value            */
         break;
       case 2:
         varBind_p = mkDPIset(               /* Make DPI set packet  */
                     varBind_p,              /* ptr to varBind chain */
                     pack_p->group_p,        /* ptr to subtree       */
                     DPI_SIMPLE_STRING,      /* ptr to rest of OID   */
                     SNMP_TYPE_DisplayString,/* value type           */
                     strlen(value2_p),
                     value2_p);              /* ptr to value         */
         break;
       case 3:
         varBind_p = mkDPIset(               /* Make DPI set packet  */
                     varBind_p,              /* ptr to varBind chain */
                     pack_p->group_p,        /* ptr to subtree       */
                     DPI_SIMPLE_OID,         /* ptr to rest of OID   */
                     SNMP_TYPE_OBJECT_IDENTIFIER,/* value type       */
                     strlen(value3_p),
                     value3_p);              /* ptr to value         */
         break;
       case 4:
         varBind_p = mkDPIset(               /* Make DPI set packet  */
                     varBind_p,              /* ptr to varBind chain */
```

```
                pack_p->group_p,        /* ptr to subtree       */
                DPI_SIMPLE_NULL,        /* ptr to rest of OID   */
                SNMP_TYPE_NULL,         /* value type           */
                strlen(value4_p),
                value4_p);              /* ptr to value         */
      break;
    case 5:
      varBind_p = mkDPIset(             /* Make DPI set packet  */
                varBind_p,              /* ptr to varBind chain */
                pack_p->group_p,        /* ptr to subtree       */
                DPI_SIMPLE_IPADDRESS,   /* ptr to rest of OID   */
                SNMP_TYPE_IpAddress,    /* value type           */
                sizeof(value5),         /* length of value      */
                &value5);               /* ptr to value         */
      break;
    case 6:
      varBind_p = mkDPIset(             /* Make DPI set packet  */
                varBind_p,              /* ptr to varBind chain */
                pack_p->group_p,        /* ptr to subtree       */
                DPI_SIMPLE_COUNTER,     /* ptr to rest of OID   */
                SNMP_TYPE_Counter32,    /* value type           */
                sizeof(value6),         /* length of value      */
                &value6);               /* ptr to value         */
      break;
    case 7:
      varBind_p = mkDPIset(             /* Make DPI set packet  */
                varBind_p,              /* ptr to varBind chain */
                pack_p->group_p,        /* ptr to subtree       */
                DPI_SIMPLE_GAUGE,       /* ptr to rest of OID   */
                SNMP_TYPE_Gauge32,      /* value type           */
                sizeof(value7),         /* length of value      */
                &value7);               /* ptr to value         */
      break;
    case 8:
      varBind_p = mkDPIset(             /* Make DPI set packet  */
                varBind_p,              /* ptr to varBind chain */
                pack_p->group_p,        /* ptr to subtree       */
                DPI_SIMPLE_TIMETICKS,   /* ptr to rest of OID   */
                SNMP_TYPE_TimeTicks,    /* value type           */
                sizeof(value8),         /* length of value      */
                &value8);               /* ptr to value         */
      break;
    case 9:
      varBind_p = mkDPIset(             /* Make DPI set packet  */
                varBind_p,              /* ptr to varBind chain */
                pack_p->group_p,        /* ptr to subtree       */
                DPI_SIMPLE_OPAQUE,      /* ptr to rest of OID   */
                SNMP_TYPE_Opaque,       /* value type           */
                strlen(value9_p),
                value9_p);              /* ptr to value         */
      break;




   default:
      varBind_p = mkDPIset(             /* Make DPI set packet  */
                varBind_p,              /* ptr to varBind chain */
                pack_p->group_p,        /* ptr to subtree       */
                pack_p->instance_p,     /* ptr to rest of OID   */
                SNMP_TYPE_endOfMibView, /* value type           */
                0L,                     /* length of value      */
                (unsigned char *)0);    /* ptr to value         */
      break;
    } /* endswitch */


   varBind_p = mkDPIset( varBind_p, pack_p->group_p,
                         pack_p->instance_p,
                         SNMP_TYPE_endOfMibView,
                         0L,
                         (unsigned char *)0           );


pack_p = pack_p->next_p; /* to process any other varbinds */
} while( pack_p != NULL );


if (!varBind_p) return(-2); /* something is probably wrong
                               with this subagent code. shut
```

```
                              it down.
                            */

   packet_p = mkDPIresponse(          /* Make DPIresponse packet */
                   hdr_p,                /* ptr parsed request      */
                   SNMP_ERROR_noError,   /* all is OK, no error     */
                   0L,                   /* index is zero, no error */
                   varBind_p);           /* varBind response data   */

   if (!packet_p) return(-2);         /* If it failed, return    */



   /*-------------------------------------------------------------
   */
   if ((rc=sendDPIpacket(packet_p,DPI_PACKET_LEN(packet_p)))
                               !=snmpsa_RC_ok) {
       printf("send failed in handle_next()\n");
       return rc;
       }


   return(rc);
   } /* handle_next() */
```

```
/*********************************************************************/
/* Processing a SET/COMMIT/UNDO request                              */
/*********************************************************************/
/*                                                                   */
/* These 3 requests can come in one of these sequences:              */
/*    - SET, COMMIT                                                  */
/*    - SET, UNDO                                                    */
/*    - SET, COMMIT, UNDO                                            */
/* Normal sequence is SET and then COMMIT. When we receive a SET     */
/* request, we must make preparations to accept the new value        */
/* like check that it is for an existing object and instance, check  */
/* the value type and contents to be valid, allocate memory etc),    */
/* but we must not yet effectuate the change.                        */
/* If all goes well, the next request we receive will be a COMMIT    */
/* request. It is then that we must effectuate the change, but we    */
/* must then also keep enough information such that we can UNDO      */
/* the change later if we get a subsequent UNDO request. The latter  */
/* may happen if the agent discovers any errors with other           */
/* subagents while processing requests that belong to the same       */
/* original SNMP SET packet (all the varBinds in the same SNMP       */
/* request PDU must be processed "as if atomic").                    */
/*                                                                   */
/* When the DPI packet is parsed, the snmp_dpi_hdr structure shows   */
/* in the packet_type that this is an SNMP_DPI_SET, SNMP_DPI_COMMIT  */
/* SNMP_DPI_UNDO packet. In that case, the data_u field contains a   */
/* ptr to a SET-varBind, represented in an snmp_dpi_get_packet       */
/* structure (COMMIT and UNDO have same varBind data as SET upon     */
/* which they follow):                                               */
/*                                                                   */
/*   struct dpi_set_packet {                                         */
/*     char                  *object_p;    / * ptr to OIDstring     */
/*     char                  *group_p;     / * ptr to subtree       */
/*     char                  *instance_p;  / * ptr to rest of OID   */
/*     unsigned char          value_type;  / * SNMP_TYPE_xxxx       */
/*     unsigned short         value_len;   / * value length         */
/*     char                  *value_p;     / * ptr to value itself  */
/*     struct dpi_set_packet  *next_p;      / * ptr to next in chain */
/*   };                                                              */
/*   typedef struct dpi_set_packet        snmp_dpi_set_packet;       */
/*   #define snmp_dpi_set_packet_NULL_p   ((snmp_dpi_set_packet *)0) */
/*                                                                   */
/* So, assuming we have registered example subtree dpiSimpleMIB      */
/* and a GET request comes in for one variable dpiSimpleString.0     */
/* (so that is object 1 instance 0 in our subtree), and also         */
/* assuming that the agent knows about our compiled dpiSimpleMIB     */
/* so that it knows this is a DisplayString as opposed to just an    */
/* arbitrary OCTET_STRING, then the ptrs in the snmp_dpi_set_packet  */
/* structure would have ptrs and values like:                        */
/*                                                                   */
/*   object_p    ->  "1.3.6.1.4.1.2.2.1.5.2.0"                       */
/*   group_p     ->  "1.3.6.1.4.1.2.2.1.5"                           */
/*   instance_p  ->  "2.0"                                           */
/*   value_type  ->  SNMP_TYPE_DisplayString                         */
/*   value_len   ->  8                                               */
```

```
/*   value_p     ->  ptr to the value to be set.                    */
/*   next_p      ->  snmp_dpi_get_packet_NULL_p                     */
/*                                                                  */
/* If there are multiple varBinds in a SET request, then each one   */
/* is represented in a snmp_dpi_set_packet structure and all the    */
/* snmp_dpi_set_packet structures are chained via the next ptr.     */
/* As long as the next ptr is not the snmp_dpi_set_packet_NULL_p    */
/* pointer then there are more varBinds in the list.                */
/*                                                                  */
/* Now we can analyze the varBind structure for whatever checking we  */
/* want to do. Once we are ready to make a response that contains   */
/* the value of the variable, we may prepare a new SET-varBind.     */
/* However, by definition, the response to a successful SET is      */
/* exactly the same as the SET request. So there is no need to      */
/* return any varBinds. A simple response with SNMP_ERROR_noError   */
/* and an index of zero will do. In case that there is an error,    */
/* then a simple response with the SNMP_ERROR_xxxx error code and   */
/* an index pointing to the varBind in error (counting starts at 1)  */
/* will do.                                                         */
/*                                                                  */
/* Here follows a code sample to return a response. We assume that  */
/* there are no errors in the request, but proper code should do    */
/* the checking for that. We also do not check if the varBind in    */
/* the COMMIT and/or UNDO is the same as that in the SET request.   */
/* A proper agent would make sure that that is the case, but a      */
/* proper subagent may want to verify that for itself.              */
/* We only do one simple check that this is dpiSimpleString.0 and if */
/* it is not we return a noCreation, which may not be correct.      */
/* The mainline does not even return a response.                    */
/********************************************************************/
static int handle_set(snmp_dpi_hdr *hdr_p,
                      snmp_dpi_set_packet *pack_p)
{
       unsigned char       *packet_p;
       int                  rc,i;
       int                  index       = 0;
       int                  error       = SNMP_ERROR_noError;
       snmp_dpi_set_packet *varBind_p;

       varBind_p =                       /* init the varBind chain  */
          snmp_dpi_set_packet_NULL_p;    /* to a NULL pointer       */



   /*-------------------------------------------------------------
   |   must handle multi-varbinds!!
   */
   do {


       if (!pack_p->instance_p ||
           (strcmp(pack_p->instance_p,"1.0")  != 0  &&
         strcmp(pack_p->instance_p,"2.0")  != 0  &&
         strcmp(pack_p->instance_p,"3.0")  != 0  &&
         strcmp(pack_p->instance_p,"4.0")  != 0  &&
         strcmp(pack_p->instance_p,"5.0")  != 0  &&
         strcmp(pack_p->instance_p,"6.0")  != 0  &&
         strcmp(pack_p->instance_p,"7.0")  != 0  &&
         strcmp(pack_p->instance_p,"8.0")  != 0  &&
         strcmp(pack_p->instance_p,"9.0")  != 0 ))
      {
         if (pack_p->instance_p &&
             (strncmp(pack_p->instance_p,"1.",2) == 0))
           {
             error = SNMP_ERROR_notWritable;
           } else if (pack_p->instance_p &&
            (strncmp(pack_p->instance_p,"2.",2) == 0))
           {
             error = SNMP_ERROR_notWritable;
           } else if (pack_p->instance_p &&
            (strncmp(pack_p->instance_p,"3.",2) == 0))
           {
             error = SNMP_ERROR_notWritable;
           } else if (pack_p->instance_p &&
            (strncmp(pack_p->instance_p,"4.",2) == 0)) {
             error = SNMP_ERROR_notWritable;
              }
             else if (pack_p->instance_p &&
           (strncmp(pack_p->instance_p,"5.",2) == 0)) {
            error = SNMP_ERROR_notWritable;
             }
             else if (pack_p->instance_p &&
```

```
        (strncmp(pack_p->instance_p,"6.",2) == 0)) {
         error = SNMP_ERROR_notWritable;
          }
          else if (pack_p->instance_p &&
        (strncmp(pack_p->instance_p,"7.",2) == 0)) {
         error = SNMP_ERROR_notWritable;
          }
          else if (pack_p->instance_p &&
        (strncmp(pack_p->instance_p,"8.",2) == 0)) {
        error = SNMP_ERROR_notWritable;
          }
          else if (pack_p->instance_p &&
        (strncmp(pack_p->instance_p,"9.",2) == 0)) {
        error = SNMP_ERROR_notWritable;
             }
          else if (pack_p->instance_p &&
        (strncmp(pack_p->instance_p,"10.",2) == 0)) {
        error = SNMP_ERROR_notWritable;
             }

       else {
        error = SNMP_ERROR_noCreation;
       } /* endif */

       packet_p = mkDPIresponse(       /* Make DPIresponse packet */
                hdr_p,                 /* ptr parsed request      */
                error,                 /* whatever                */
                1,                     /* index is 1, 1st varBind */
                varBind_p);            /* varBind response data   */

       if (!packet_p) return(-1);


       if ((rc=sendDPIpacket(packet_p,DPI_PACKET_LEN(packet_p)))
                    !=snmpsa_RC_ok) {
          /* msg */
           return rc;
           }

       return(rc);
    } /* if checking for *.0 */



/*-------------------------------------------------------------------
   here if things still look ok for set
*/
if (strcmp(pack_p->instance_p,"1.0")  == 0 ) {
    memcpy(&new_val1,pack_p->value_p,pack_p->value_len);
    }
else if (strcmp(pack_p->instance_p,"2.0")  == 0 ) {
    if (new_val2_p) free(new_val2_p);
    if (old_val2_p) free(old_val2_p);
    new_val2_p = (char*) malloc(pack_p->value_len);
                                       /* new value to set */
    if (new_val2_p) {
                                       /* If success, then also   */
        memcpy((char*)new_val2_p,      /* copy new value to our   */
             pack_p->value_p,        /* own and newly allocated */
           pack_p->value_len);     /* memory area.            */
        }
    else {                           /* Else failed to malloc,  */
     error = SNMP_ERROR_genErr;   /* so that is a genErr     */
        index = 1;                   /* at first varBind        */
        }
    }
else if (strcmp(pack_p->instance_p,"3.0")  == 0 ) {
    if (new_val3_p) free(new_val3_p);
    if (old_val3_p) free(old_val3_p);

    new_val3_p = (char*)
    malloc(pack_p->value_len);
    if (new_val3_p) {
        memcpy(new_val3_p,
               pack_p->value_p,
               pack_p->value_len);
        }
    else {
         error = SNMP_ERROR_genErr;
         index = 1;
      }
    }
else if (strcmp(pack_p->instance_p,"4.0")  == 0 ) {
```

```
        /* is null oid, nothing to do */
        }
    else if (strcmp(pack_p->instance_p,"5.0")  == 0 ) {
        memcpy(&new_val5,pack_p->value_p,pack_p->value_len);
        }
    else if (strcmp(pack_p->instance_p,"6.0")  == 0 ) {
        memcpy(&new_val6,pack_p->value_p,pack_p->value_len);
        }
    else if (strcmp(pack_p->instance_p,"7.0")  == 0 ) {
        memcpy(&new_val7,pack_p->value_p,pack_p->value_len);
        }
    else if (strcmp(pack_p->instance_p,"8.0")  == 0 ) {
        memcpy(&new_val8,pack_p->value_p,pack_p->value_len);
        }
    else if (strcmp(pack_p->instance_p,"9.0")  == 0 ) {
        if (new_val9_p) free(new_val9_p); /* free these memory areas*/
        if (old_val9_p) free(old_val9_p);  /* if we allocated any   */

        new_val9_p =  (char*)          /* allocate memory for     */
        malloc(pack_p->value_len);   /* new value to set        */
        if (new_val9_p) {                /* If success, then also   */
            memcpy(new_val9_p,             /* copy new value to our  */
                pack_p->value_p,       /* own and newly allocated */
                     pack_p->value_len);    /* memory area.          */
            }
        else {                         /* Else failed to malloc,  */
         error = SNMP_ERROR_genErr;    /* so that is a genErr     */
         index = 1;                    /* at first varBind        */
         }
        }

    else {
        error = SNMP_ERROR_noCreation;
        index = 0;
        }

    /* since multi-varbinds may exist, and because response to set
       relies on index to error varbind, lets use only generic
       varbinds in our response to the sets (assuming these are
       ignored in agent unless the index points to one, in which
       case it will be the last one.)
    */
    varBind_p = mkDPIset( varBind_p,             /* ptr to varBind chain */
                          pack_p->group_p,       /* ptr to subtree       */
                          pack_p->instance_p,    /* ptr to rest of OID   */
                          pack_p->value_type,    /* value type Integer 32*/
                          pack_p->value_len,     /* length of value      */
                          NULL );                /* ptr to value         */

    pack_p = pack_p->next_p; /* to process any other varbinds */
    } while( pack_p != NULL );



    /*-----------------------------------------------------------------
       make and send normal response
    */
    packet_p = mkDPIresponse( hdr_p, error, index, varBind_p );

    if (!packet_p) return(-1);      /* If it failed, return    */

    if ((rc = sendDPIpacket(packet_p,DPI_PACKET_LEN(packet_p)))
              != snmpsa_RC_ok) {
        /* msg */
        return rc;
        }

    return(rc);
    } /* handle_set() */



/*--------------------------------------------------------------------------
*/
static int handle_commit(snmp_dpi_hdr *hdr_p,
                    snmp_dpi_set_packet *pack_p) {

     unsigned char       *packet_p;
     int                  rc,i;
     int                  index       = 0;
     int                  error       = SNMP_ERROR_noError;
```

```
  snmp_dpi_set_packet *varBind_p;

varBind_p = snmp_dpi_set_packet_NULL_p;



/*-------------------------------------------------------------
|   loop to handle multi-varbinds
*/
do {


if (strcmp(pack_p->instance_p,"1.0")  == 0 ) {
    old_val1 = cur_val1;
    cur_val1 = new_val1;
    new_val1 = 0;
    }
else if (strcmp(pack_p->instance_p,"2.0")  == 0 ) {
    old_val2_p = cur_val2_p;           /* save old value for undo */
    cur_val2_p = new_val2_p;           /* make new value current  */

    new_val2_p = (char *)0;           /* keep only 1 ptr around  */
    /* may need to convert from ASCII to native if OCTET_STRING */
    }
else if (strcmp(pack_p->instance_p,"3.0")  == 0 ) {
    old_val3_p = cur_val3_p;           /* save old value for undo */
    cur_val3_p = new_val3_p;           /* make new value current  */
    new_val3_p = (char *)0;           /* keep only 1 ptr around  */
    }
else if (strcmp(pack_p->instance_p,"4.0")  == 0 ) {
    /* is null oid, nothing to do */
    }
else if (strcmp(pack_p->instance_p,"5.0")  == 0 ) {
    old_val5 = cur_val5;
    cur_val5 = new_val5;
    new_val5 = 0;
    }
else if (strcmp(pack_p->instance_p,"6.0")  == 0 ) {
    old_val6 = cur_val6;
    cur_val6 = new_val6;
    new_val6 = 0;
    }
else if (strcmp(pack_p->instance_p,"7.0")  == 0 ) {
    old_val7 = cur_val7;
    cur_val7 = new_val7;
    new_val7 = 0;
    }
else if (strcmp(pack_p->instance_p,"8.0")  == 0 ) {
    old_val8 = cur_val8;
    cur_val8 = new_val8;
    new_val8 = 0;
    }
else if (strcmp(pack_p->instance_p,"9.0")  == 0 ) {
    old_val9_p = cur_val9_p;           /* save old value for undo */
    cur_val9_p = new_val9_p;           /* make new value current  */
    new_val9_p = (char *)0;           /* keep only 1 ptr around  */
    }


else {
    error = SNMP_ERROR_noCreation;
    index = 0;
    }


pack_p = pack_p->next_p; /* to process any other varbinds */
} while( pack_p != NULL );



/*-------------------------------------------------------------
   make and send normal response
*/
packet_p = mkDPIresponse( hdr_p, error, index, varBind_p );
if (!packet_p) return(-1);      /* If it failed, return    */
if ((rc=sendDPIpacket(packet_p,DPI_PACKET_LEN(packet_p)))
                        !=snmpsa_RC_ok) {
    /* msg */
    return rc;
    }
```

```
   if ((rc=sendDPIpacket( packet_p,DPI_PACKET_LEN(packet_p)))
                               !=snmpsa_RC_ok) {
       /* msg */
       return rc;
       }


   return(rc);
   } /*handle_commit()*/



/*-------------------------------------------------------------------------
*/
static int handle_undo(snmp_dpi_hdr *hdr_p,
                    snmp_dpi_set_packet *pack_p) {

     unsigned char       *packet_p;
     int                  rc,i;
     int                  index       = 0;
     int                  error       = SNMP_ERROR_noError;
     snmp_dpi_set_packet *varBind_p;

   varBind_p = snmp_dpi_set_packet_NULL_p;


   /*-----------------------------------------------------------------
   |   must handle multi-varbinds!!
   */
   do {


   if (strcmp(pack_p->instance_p,"1.0")  == 0 ) {
       cur_val1 = old_val1;
       }
   else if (strcmp(pack_p->instance_p,"2.0")  == 0 ) {
       if (new_val2_p) free(new_val2_p);  /* free allocated memory   */
       cur_val2_p = old_val2_p;           /* reset to old value      */
       }
   else if (strcmp(pack_p->instance_p,"3.0")  == 0 ) {
       if (new_val3_p) free(new_val3_p);  /* free allocated memory   */
       cur_val2_p = old_val2_p;           /* reset to old value      */
       }
   else if (strcmp(pack_p->instance_p,"4.0")  == 0 ) {
       /* null oid */
       }
   else if (strcmp(pack_p->instance_p,"5.0")  == 0 ) {
       cur_val5 = old_val5;
       }
   else if (strcmp(pack_p->instance_p,"6.0")  == 0 ) {
       cur_val6 = old_val6;
       }
   else if (strcmp(pack_p->instance_p,"7.0")  == 0 ) {
       cur_val7 = old_val7;
       }
   else if (strcmp(pack_p->instance_p,"8.0")  == 0 ) {
       cur_val8 = old_val8;
       }
   else if (strcmp(pack_p->instance_p,"9.0")  == 0 ) {
       if (new_val9_p) free(new_val9_p);  /* free allocated memory   */
       cur_val9_p = old_val9_p;           /* reset to old value      */
       }


   else {
       error = SNMP_ERROR_noCreation;
       index = 0;
       }


   pack_p = pack_p->next_p; /* to process any other varbinds */
   } while( pack_p != NULL );



   /*-------------------------------------------------------------           make response packet and send it
   */
   packet_p = mkDPIresponse( hdr_p, error, index, varBind_p );

   if (!packet_p) return(-1);      /* If it failed, return    */
```

```
   if ((rc=sendDPIpacket(packet_p,DPI_PACKET_LEN(packet_p)))
                          !=snmpsa_RC_ok) {
       /* msg */
       return rc;
       }

   return(rc);

   } /*handle_undo()*/



/*********************************************************************/
/* Function to handle a DPI UNREGISTER request                       */
/*********************************************************************/
/* An agent can send an UNREGISTER packet if some other subagent does*/
/* a register for the same subtree at a higher priority. In this case*/
/* we can decide to keep the connection open, we may regain control  */
/* over the subtree if that higher priority registration goes away.  */
/* An agent can also send an UNREGISTER if for instance an SNMP      */
/* manager tells it to "invalidate" the subagent connection or the   */
/* registered subtree. In this case we decide to give up.            */
/*                                                                   */
/* Here is a very simple sample piece of code to handle such a packet*/
/*********************************************************************/
static int handle_unreg( snmp_dpi_hdr *hdr_p,
                         snmp_dpi_ureg_packet *pack_p) {
   /* msg */
   if (pack_p->reason_code ==
                    SNMP_UNREGISTER_higherPriorityRegistered) {
       return(0); /* keep waiting, we may regain subtree later */
       }

   return(-1); /* causes exit in main loop */
   } /* handle_unreg() */



/*********************************************************************/
/* Function to handle a DPI CLOSE request                            */
/*********************************************************************/
/* An agent can send a CLOSE packet if it encounters an error or for */
/* some other reason. It can also do so if an SNMP MANAGER tells it  */
/* to "invalidate" the subagent connection.                          */
/*                                                                   */
/* Here is a very simple sample piece of code to handle such a packet*/
/*********************************************************************/
static int handle_close(snmp_dpi_hdr *hdr_p,
                         snmp_dpi_close_packet *pack_p){

   /* msg */
   return(-1); /* causes exit in main loop */
   } /* handle_close() */



/*--------------------------------------------------------------------
*/
static int handle_resp(snmp_dpi_hdr *hdr_p, snmp_dpi_resp_packet *pack_p)
   {
   /* msg */
   return(0);
   } /* handle_resp() */



/* ----------------------------------------- end of SAsample.c ---- */
```

Chapter 8. OS/400 SNMP-Based Functions

This chapter describes some AS/400 functions and applications which use the SNMP support provided by OS/400.

8.1 Client Inventory Management

Client Inventory Management is a built-in function of OS/400 which uses the OS/400 SNMP support. SNMP is the mechanism through which OS/400 gathers information about personal computer clients that are attached to the AS/400. The information is stored in AS/400 database files to provide a comprehensive inventory of personal computer assets. An example of the type of information available is the following:

- **System** - connectivity, access, and management information
- **Hardware** - disk drive, memory, hard file, etc
- **Software** - installed software identification and change information

If a client is configured to send SNMP traps to a managing AS/400, then the client's information is collected automatically through a process known as *automatic discovery*. Client Access/400 Optimized for OS/2 provides the SNMP support required for automatic discovery. To receive and process the incoming traps from the client, the OS/400 SNMP agent and Trap Manager must be active on the managing AS/400.

The client system does not necessarily have to be a Client Access/400 Optimized for OS/2 system. Any client system that supports SNMP with automatic discovery can be managed. To provide as much information as possible, the client should support MIB II, the host resources MIB, APPN MIB and DMI MIB.

Traps received by an AS/400 system are processed differently for *new* clients (clients that have not been connected to the AS/400 before) than traps received for existing clients. Traps received from a new client are processed immediately and the client's information is retrieved and stored in the AS/400 databases. For traps received from existing clients, hardware and software information is refreshed only if it is more than 30 days old. 30 days is the default value. The default value can be changed to any value between 1 and 365 days. Any other value will default to 30 days. A data area QZCAREFI in library QUSRSYS should be created to change the refresh interval. For example:

```
CRTDTAARA DTAARA(QUSRSYS/QZCAREFI)
  TYPE(*DEC) LEN(2 0) VALUE(75)
  TEXT('Refresh Interval')
```

This example sets the refresh interval at 75 days. The data area must be of type decimal (integer value). Using this example, a refresh interval of 75 days is used for all clients that the AS/400 system is managing.

To change the refresh interval to a three digit value having created a data area with a two digit value, you must first delete the existing data area and then create a new one specifying a three digit value.

If necessary, the database records associated with a client system can be erased. To do this, end the SNMP server job and call program QZCARMVC using the appropriate parameters as shown in the example below.

```
ENDTCPSVR  SERVER(*SNMP)
CALL QZCARMVC PARM('***ZCAIDX***' X'00000000')
```

Where ***ZCAIDX*** is the unique index associated with the client in the AS/400 database. See the 8.1.1, “Client Software Management Database Formats” on page 129 for more information on ZCAIDX.

For more details about this task refer to *Inside Client Access/400 Optimized for OS/2*, SG24-2587.

By accessing the client management databases, users can write application programs that perform resource, asset, license, and network management functions. The client nanagement databases, which are a set of physical and logical files, represent the client information that was gathered using SNMP GET and GETNEXT requests when the client was connected to the AS/400. The database structure is based on the MIB objects that are being shadowed from the client.

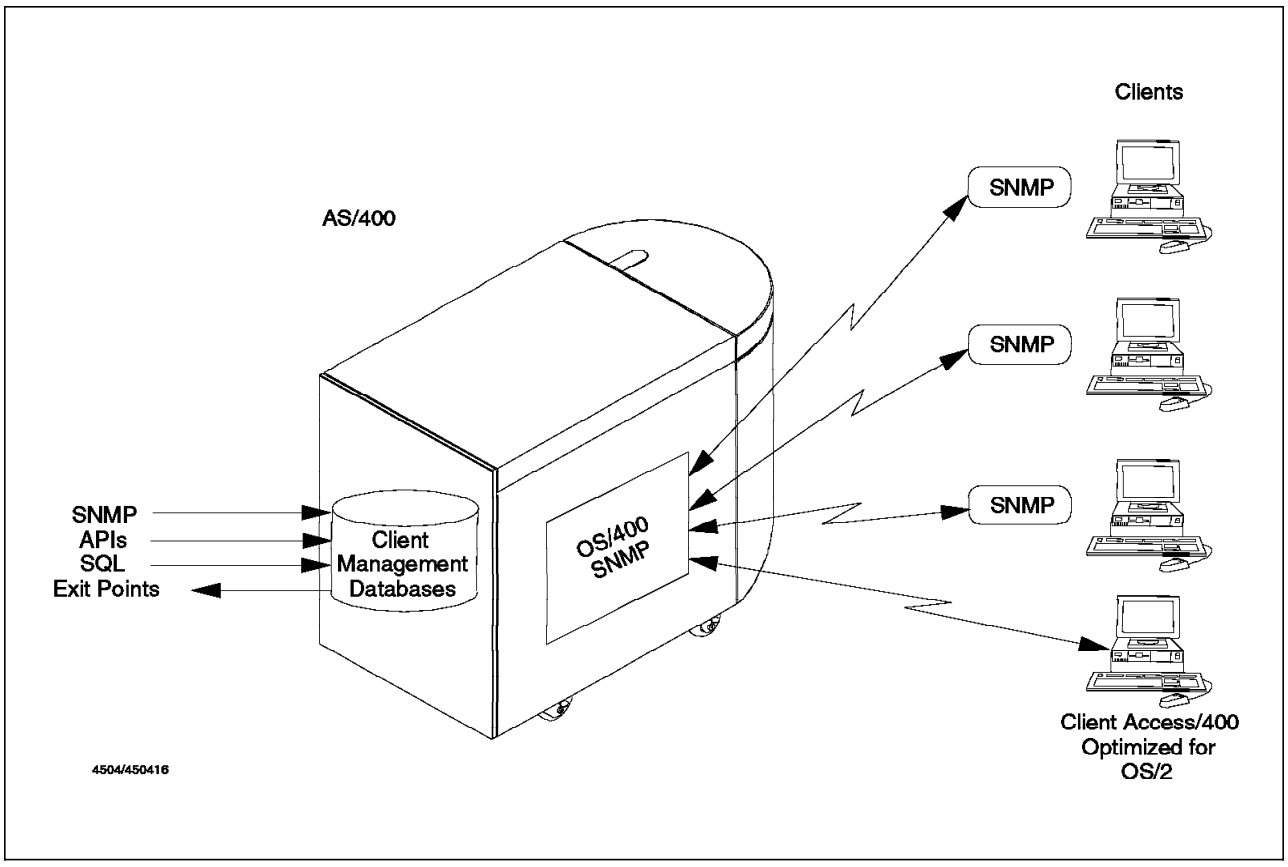


Figure 84. Client Inventory Management Overview

The client management database information can also be retrieved from an SNMP manager via the Client Management MIB.

The Client Management MIB is an IBM enterprise-specific MIB module shipped with OS/400. SNMP managing applications such as NetView for OS/2 or NetView for AIX must load the MIB module before it can be used to perform SNMP

operations. Chapter 13, “Loading an Enterprise-Specific MIB” on page 219 explains the procedure for loading an enterprise-specific MIB module into NetView for OS/2. On the AS/400, the Client Management MIB is member IBMCLTM in file QSYS/QANMMIB.

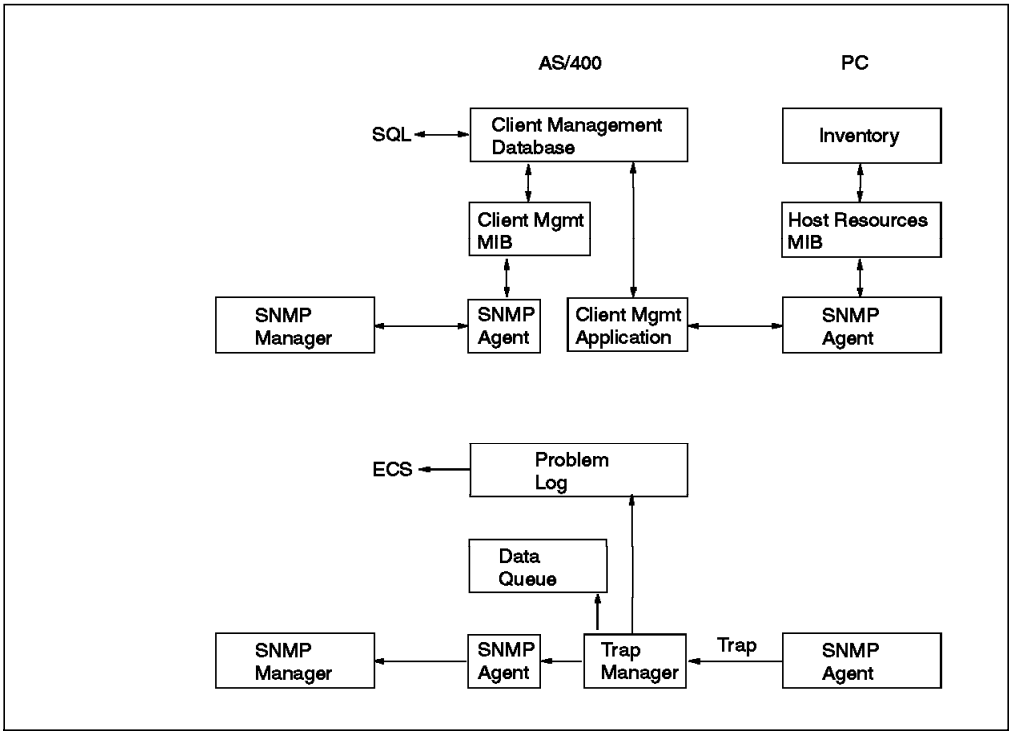


Figure 85. Client Management Function Overview

Figure 85 shows how the Client Management MIB can be used via an SNMP manager to retrieve information from the client management database on the AS/400. The client management database information having previously been retrieved from the managed clients via the Host Resources MIB supported by Client Access/400 Optimized for OS/2 and other PC SNMP agent implementations. The figure also shows how traps received from the managed clients can either be handled locally or forwarded to an SNMP manager via the forward trap (*YES) STRTRPMGR command option.

8.1.1 Client Software Management Database Formats

The following are the database formats used when querying and writing applications that access client management information. All of these database files are shipped with OS/400 in library QSYS. The files are copied into the QUSRSYS library when the first client is discovered and stored in the database files. All client information is stored in the database files in the QUSRSYS library. The files in the QSYS library are for recovery purposes. Client management applications can query these client databases, including user-written SQL applications.

The format of each of the Client Management databases are presented in the following sections.

8.1.1.1 QAZCADEV

```
***   Start of specifications   *********************************
*
*  File Name:  qazcadev
*  File Type:  physical
*
*  Function:  Database file where the client device
*             information is maintained.  The data is obtained
*             from the HOST MIB.
*
*         ZCAIDX --> Client Index
*         ZCADEV --> Device Index
*         ZCATYP --> Device Type
*         ZCADID --> Device ID
*         ZCASTT --> Device Status
*         ZCAERR --> Device Errors
*         ZCADSC --> Device Description
*
***   End of Specifications   *********************************
                                    UNIQUE
          R QZCADEVR
            ZCAIDX        12H
            ZCADEV         9B
            ZCATYP         9B
            ZCADID       128A         VARLEN(128)
            ZCASTT         9B
            ZCAERR         9B
            ZCADSC        64A         VARLEN(64)
          K ZCAIDX
          K ZCADEV
```

8.1.1.2 QAZCADIR

```
***   Start of specifications   *********************************
*
*  File Name:  qazcadir
*  File Type:  physical
*
*  Function:  Database file where the basic client directory
*             information is maintained.
*
*         ZCBIDX --> Client Index
*         ZCBCLT --> Client ID
*         ZCBDSC --> Client Description
*         ZCBCMN --> Community
*         ZCBIPA --> IP Address
*         ZCBCPN --> CP NetID
*
***   End of Specifications   *********************************
                                    UNIQUE
          R QZCADIRR
            ZCBIDX        12H
            ZCBCLT       255A         VARLEN(255)
            ZCBDSC       255A         VARLEN(255)
            ZCBCMN       255A         VARLEN(255)
            ZCBIPA        15A         VARLEN(15)
            ZCBCPN        17A         VARLEN(17)
          K ZCBIDX
```

8.1.1.3 QAZCADRL

```
***   Start of specifications   *********************************
*
*  File Name:  qazcadrl
*  File Type:  logical
*
*  Function:  Database file where the basic client directory
*             information is maintained
*
*         ZCBIDX --> Client index
*         ZCBCLT --> Client ID
*         ZCBDSC --> Client Description
*         ZCBCMN --> Community
*         ZCBIPA --> IP Address
*         ZCBCPN --> CP NetID
*
***   End of Specifications   *********************************
                                    UNIQUE

          R QZCADRLR                PFILE(QSYS/QAZCADIR)
            ZCBCLT       255A         VARLEN
            ZCBIDX        12H
```

```
            ZCBDSC       255A         VARLEN
            ZCBCMN       255A         VARLEN
            ZCBIPA        15A         VARLEN
            ZCBCPN        17A         VARLEN
          K ZCBCLT
```

8.1.1.4 QAZCADSK

```
***   Start of specifications   *********************************
*
*  File Name:  qazcadsk
*  File Type:  physical
*
*  Function:  Database file where the client disk storage
*             information is maintained.  The data is obtained
*             from the HOST MIB.
*
*         ZCCIDX --> Client Index
*         ZCCDEV --> Device Index
*         ZCCACC --> Disk Access
*         ZCCMED --> Disk Media
*         ZCCRMV --> Disk Media Remove
*         ZCCCAP --> Disk Capacity
*
***   End of Specifications   *********************************
                                      UNIQUE
          R QZCADSKR
            ZCCIDX        12H
            ZCCDEV         9B
            ZCCACC         9B
            ZCCMED         9B
            ZCCRMV         9B
            ZCCCAP         9B
          K ZCCIDX
          K ZCCDEV
```

8.1.1.5 QAZCAFS

```
***   Start of specifications   *********************************
*
*  File Name:  qazcafs
*  File Type:  physical
*
*  Function:  Database file where the client file system
*             information is maintained.  The data is obtained
*             from the HOST MIB.
*
*         ZCDIDX --> Client Index
*         ZCDFS  --> File System Index
*         ZCDLMP --> Mount Point
*         ZCDRMP --> Remote Mount
*         ZCDTYP --> Type
*         ZCDACC --> Access
*         ZCDBOT --> Bootable
*         ZCDSTG --> Storage Index
*         ZCDFBK --> Full Backup
*         ZCDPBK --> Partial Backup
*
***   End of Specifications   *********************************
                                      UNIQUE
          R QZCAFSR
            ZCDIDX        12H
            ZCDFS          9B
            ZCDLMP       128A         VARLEN(128)
            ZCDRMP       128A         VARLEN(128)
            ZCDTYP         9B
            ZCDACC         9B
            ZCDBOT         9B
            ZCDSTG         9B
            ZCDFBK          Z
            ZCDPBK          Z
          K ZCDIDX
          K ZCDFS
```

8.1.1.6 QAZCAMSC

```
***   Start of specifications   *********************************
*
*  File Name:  qazcamsc
*  File Type:  physical
*
*  Function:  Database file where the basic client directory
*             information is maintained
*
*         ZCBIDX --> Client Index
*         ZCBMEM --> Memory size
*         ZCBUPT --> Up Time
*         ZCBSTT --> System Status
*         ZCBMBI --> Support MIBII
*         ZCBMBH --> Support HOST MIB
*         ZCBMBA --> Support APPN MIB
*         ZCBMBX --> Support Extensions
*         ZCBHDW --> Hardware Refresh
*         ZCBSFW --> Software Refresh
*         ZCBCON --> Contact
*         ZCBLOC --> Location
*         ZCBTYP --> Machine Type
*         ZCBMDL --> Machine Model
*         ZCBUSR --> User Profile
*         ZCBOWN --> Owner
*         ZCBPHN --> Owner Phone
*         ZCBOFC --> Office
*
***   End of Specifications   *********************************
                                      UNIQUE
          R QZCAMSCR
            ZCBIDX        12H
            ZCBMEM         9B
            ZCBUPT         9B
            ZCBSTT         9B
            ZCBMBI         9B
            ZCBMBH         9B
            ZCBMBA         9B
            ZCBMBX         9B
            ZCBHDW          Z
            ZCBSFW          Z
            ZCBCON       255A         VARLEN(255)
            ZCBLOC       255A         VARLEN(255)
            ZCBTYP         4A         VARLEN(4)
            ZCBMDL         3A         VARLEN(3)
            ZCBUSR        10A         VARLEN(10)
            ZCBOWN        32A         VARLEN(32)
            ZCBPHN        32A         VARLEN(32)
            ZCBOFC        32A         VARLEN(32)
          K ZCBIDX
```

8.1.1.7 QAZCANET

```
***   Start of specifications   *********************************
*
*  File Name:  qazcanet
*  File Type:  physical
*
*  Function:  Database file where the client network
*             information is maintained.  The data is obtained
*             from the HOST MIB.
*
*         ZCEIDX --> Client Index
*         ZCEDEV --> Device Index
*         ZCENIF --> Network IFIndex
*
***   End of Specifications   *********************************
                                      UNIQUE
          R QZCANETR
            ZCEIDX        12H
            ZCEDEV         9B
            ZCENIF         9B
          K ZCEIDX
          K ZCEDEV
```

8.1.1.8 QAZCAPRC

```
***   Start of specifications   *********************************
*
*  File Name:  qazcaprc
*  File Type:  physical
*
*  Function:  Database file where the client processor
*             information is maintained.  The data is obtained
*             from the HOST MIB.
*
*         ZCFIDX --> Client Index
*         ZCFDEV --> Device Index
*         ZCFLOD --> Processor Load
*         ZCFFRM --> Processor Licensed Internal Code
*
***   End of Specifications   *********************************
                                       UNIQUE
          R QZCAPRCR
            ZCFIDX        12H
            ZCFDEV         9B
            ZCFLOD         9B
            ZCFFRM       128A         VARLEN(128)
          K ZCFIDX
          K ZCFDEV
```

8.1.1.9 QAZCAPRT

```
***   Start of specifications   *********************************
*
*  File Name:  qazcaprt
*  File Type:  physical
*
*  Function:  Database file where the client printer
*             information is maintained.  The data is obtained
*             from the HOST MIB.
*
*         ZCGIDX --> Client Index
*         ZCGDEV --> Device Index
*         ZCGSTT --> Status
*         ZCGERR --> Error State
*
***   End of Specifications   *********************************
                                       UNIQUE
          R QZCAPRTR
            ZCGIDX        12H
            ZCGDEV         9B
            ZCGSTT         9B
            ZCGERR         4H         VARLEN(1)
          K ZCGIDX
          K ZCGDEV
```

8.1.1.10 QAZCAPTN

```
***   Start of specifications   *********************************
*
*  File Name:  qazcaptn
*  File Type:  physical
*
*  Function:  Database file where the client storage partition
*             information is maintained.  The data is obtained
*             from the HOST MIB.
*
*         ZCHIDX --> Client Index
*         ZCHDEV --> Device Index
*         ZCHPTN --> Partition Index
*         ZCHSIZ --> Size
*         ZCHFSX --> FS Index
*         ZCHLBL --> Label
*         ZCHID  --> Id
*
***   End of Specifications   *********************************
                                       UNIQUE
          R QZCAPTNR
            ZCHIDX        12H
            ZCHDEV         9B
            ZCHPTN         9B
            ZCHSIZ         9B
            ZCHFSX         9B
            ZCHLBL       128A         VARLEN(128)
            ZCHID        128H         VARLEN(128)
          K ZCHIDX
          K ZCHDEV
```

```
          K ZCHPTN
```

8.1.1.11 QAZCASFW

```
***   Start of specifications   *********************************
*
*  File Name:  qazcasfw
*  File Type:  physical
*
*  Function:  Database file where the client software
*             information is maintained.  The data is obtained
*             from the HOST MIB extensions.
*
*         ZCJIDX --> Client Index
*         ZCJSFW --> Software Index
*         ZCJTYP --> Software Type
*         ZCJSTT --> Software Status
*         ZCJID  --> Software Id
*         ZCJVER --> Software Version
*         ZCJOPT --> Software Option
*         ZCJFTR --> Software Feature
*         ZCJMNF --> Software Manufacture
*         ZCJPTH --> Software Path
*         ZCJDAT --> Software Date
*         ZCJNAM --> Software Name
*         ZCJSN  --> Software SerialNumber
*
***   End of Specifications   *********************************
                                       UNIQUE
          R QZCASFWR
            ZCJIDX        12H
            ZCJSFW         9B
            ZCJTYP         9B
            ZCJSTT         9B
            ZCJID        128A         VARLEN(7)
            ZCJVER        64A         VARLEN(6)
            ZCJOPT        16A         VARLEN(4)
            ZCJFTR        16A         VARLEN(4)
            ZCJMNF        64A         VARLEN(32)
            ZCJPTH       255A         VARLEN(128)
            ZCJDAT          Z
            ZCJNAM        64A         VARLEN(32)
            ZCJSN         64A         VARLEN(32)
          K ZCJIDX
          K ZCJSFW
```

8.1.1.12 QAZCASFX

```
***   Start of specifications   *********************************
*
*  File Name:  qazcasfx
*  File Type:  physical
*
*  Function:  Database file where the Client software fix
*             information is maintained.  The data is obtained
*             from the HOST MIB extensions.
*
*         ZCKIDX --> Client Index
*         ZCKSFW --> Software Index
*         ZCKSFX --> Software Fix Index
*         ZCKFIX --> Software Fix Id
*
***   End of Specifications   *********************************
                                       UNIQUE
          R QZCASFXR
            ZCKIDX        12H
            ZCKSFW         9B
            ZCKSFX         9B
            ZCKFIX        16A         VARLEN(7)
          K ZCKIDX
          K ZCKSFW
          K ZCKFIX
```

8.1.1.13 QAZCASTG

```
***   Start of specifications   *********************************
*
*  File Name:  qazcastg
*  File Type:  physical
*
*  Function:  Database file where the client storage
*             information is maintained.  The data is obtained
*             from the HOST MIB.
*
*         ZCIIDX --> Client index
*         ZCISTG --> Storage index
*         ZCITYP --> Type
*         ZCIDSC --> Description
*         ZCIALU --> Allocation Units
*         ZCISIZ --> Size
*         ZCIUSD --> Used
*         ZCIALF --> Allocation Failure
*
***   End of Specifications   *********************************
                                      UNIQUE
          R QZCASTGR
            ZCIIDX        12H
            ZCISTG         9B
            ZCITYP         9B
            ZCIDSC       128A         VARLEN(128)
            ZCIALU         9B
            ZCISIZ         9B
            ZCIUSD         9B
            ZCIALF         9B
          K ZCIIDX
          K ZCISTG
```

Figure 87 on page 136 and Figure 88 on page 137 show a Client Access/400 Optimized for OS/2 client's information in two different Client Management databases. The two example screens are for the QAZCADIR and QAZCAMSC database files. Once the client's information has been retrieved by the managing AS/400 system, you can view the information in the databases by running a simple AS/400 query command. For example, at the AS/400 command line, type in the following command and press PF4 the Prompt key:

```
RUNQRY QRY(*NONE)
```

Once on the screen shown in Figure 86 on page 136, fill in the File, and Library (QUSRSYS) fields under Query file: with the appropriate information for the database you wish to view, and press the Enter key.

```
                              Run Query (RUNQRY)

Type choices, press Enter.

Query  . . . . . . . . . . . . .                Name, *NONE
  Library  . . . . . . . . . . .     *LIBL       Name, *LIBL, *CURLIB
Query file:
  File . . . . . . . . . . . . .                Name, *SAME
    Library  . . . . . . . . . .     *LIBL       Name, *RUNOPT, *LIBL, *CURLIB
  Member . . . . . . . . . . . .   *FIRST        Name, *RUNOPT, *FIRST, *LAST
             + for more values
Report output type . . . . . . .   *RUNOPT       *RUNOPT, *DISPLAY...
Output form  . . . . . . . . . .   *RUNOPT       *RUNOPT, *DETAIL, *SUMMARY
Record selection . . . . . . . .   *NO           *NO, *YES









                                                                        Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 86. Prompt Screen for the RUNQRY Command

For the QAZCADIR file, you should see a screen similar to the one shown in Figure 87. Notice how the ZCBIDX, and ZCBCLT fields contain information about the client.

```
                                                          Display Report

Position to line  . . . . .
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....
       ZCBIDX        ZCBCLT
000001 asH515900470  CATCPIP
****** ********  End of report  ********









F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split
```

Figure 87. A Partial View of the QAZCADIR Database Client Information

For the QAZCAMSC file, you should see a screen similar to the one shown in Figure 88 on page 137. Notice how the information contained in this file corresponds to the fields defined in the QAZCAMSC database file format described in 8.1.1, “Client Software Management Database Formats” on page 129.

```
                                                          Display Report

Position to line  . . . . .
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....
       ZCBIDX             ZCBMEM        ZCBUPT        ZCBSTT        ZCB
000001 asH515900470       32,372        32,543             0
****** ********  End of report  ********









F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split
```

Figure 88. A Partial View of the QAZCAMSC Database Client Information

8.1.2 The Client Management MIB

The Client Management MIB is an IBM enterprise MIB that can be used to retrieve information from the AS/400 client management databases. Thus the subtree OBJECT IDENTIFIER for the Client Management MIB is located under the private subtree. The OBJECT IDENTIFER is
iso.org.dod.internet.private.enterprises.ibm.ibmprod.clientMgmtSubAgent
or, in concise form: 1.3.6.1.4.1.2.6.50

The clientMgmtSubAgent subtree has three children which form the following groups:

- clntSystem (clientMgmtSubAgent 1)
- clntHardware (clientMgmtSubAgent 2)
- clntSoftware (clientMgmtSubAgent 3)

8.1.2.1 The clntSystem Group

The Client System group comprises a table of system information about the client. Under the *clntSystemTable (clntSystem 1)*, the *clntSystemEntry (clntSystemTable 1)* object is formed by the following objects which are used for referencing the client's system information:

clntSystemIndex (clntSystemEntry 1)

clntSysName (clntSystemEntry 2)

clntSysDescr (clntSystemEntry 3)

clntCommunity (clntSystemEntry 4)

clntIpAddress (clntSystemEntry 5)

clntAppnCpNetId (clntSystemEntry 6)

clntMemorySize (clntSystemEntry 7)

clntSystemUpTime (clntSystemEntry 8)

clntSystemStatus (clntSystemEntry 9)

clntSupportMibII (clntSystemEntry 10)

clntSupportHostMib (clntSystemEntry 11)

clntSupportAppnMib (clntSystemEntry 12)

clntSupportDmiMib (clntSystemEntry 13)

clntLastHwRefresh (clntSystemEntry 14)

clntLastSwRefresh (clntSystemEntry 15)

clntSysContact (clntSystemEntry 16)

clntSysLocation (clntSystemEntry 17)

clntMachineType (clntSystemEntry 18)

clntMachineModel (clntSystemEntry 19)

clntUserProfile (clntSystemEntry 20)

clntOwner (clntSystemEntry 21)

clntOwnerPhone (clntSystemEntry 22)

clntOwnerOffice (clntSystemEntry 23)

8.1.2.2 The clntHardware Group

The Client Hardware group comprises several tables which contain objects for hardware related information about the client. Each table has its own MIB objects. For simplicity reasons here, only the hardware group tables are listed:

clntStorageTable (clntHardware 1)

clntDeviceTable (clntHardware 2)

clntProcessorTable (clntHardware 3)

clntNetworkTable (clntHardware 4)

clntPrinterTable (clntHardware 5)

clntDiskStorageTable (clntHardware 6)

clntPartitionTable (clntHardware 7)

clntFSTable (clntHardware 8)

8.1.2.3 The clntSoftware Group

The Client Software group comprises two tables which contain objects for software-related information about the client.

Under the *clntSWInstalledTable (clntSoftware 1)*, the *clntSWInstalledEntry (clntSWInstalledTable 1)* object is formed by the following objects which are used for referencing the client's installed software information:

clntSWInstalledIndex (clntSWInstalledEntry 1)

clntSWInstalledManf (clntSWInstalledEntry 2)

clntSWInstalledID (clntSWInstalledEntry 3)

clntSWInstalledVersion (clntSWInstalledEntry 4)

clntSWInstalledDateTime (clntSWInstalledEntry 5)

clntSWInstalledType (clntSWInstalledEntry 6)

clntSWInstalledPath (clntSWInstalledEntry 7)

clntSWInstalledOption (clntSWInstalledEntry 8)

clntSWInstalledLoad (clntSWInstalledEntry 9)

clntSWInstalledStatus (clntSWInstalledEntry 10)

clntSWInstalledSoftwareName (clntSWInstalledEntry 11)

clntSWInstalledSerialNumber (clntSWInstalledEntry 12)

Under the *clntSWFixTable (clntSoftware 3)*, the *clntSWFixEntry (clntSWFixTable 1)* object is formed by the following objects which are used for referencing the software fixes on the client:

clntSWFixIndex (clntSWFixEntry 1)

clntSWFixID (clntSWFixEntry 2)

8.1.3 Browsing the Client Management MIB

The following figures show some examples of the information which can be retrieved using the NetView for OS/2 MIB browser to view the clientMgmtSubAgent MIB. In order to gather the information from the AS/400 through the clientMgmtSubAgent, the client system was previously set up as a Client Access/400 Optimized for OS/2 client system configured to work over a TCP/IP network. The SNMP support and necessary software to set up this environment on the client is provided with Client Access/400 Optimized for OS/2.

Before we can query the clientMgmtSubAgent MIB objects we must load this enterprise MIB into NetView for OS/2. To do this we follow the steps shown in Chapter 13, “Loading an Enterprise-Specific MIB” on page 219. The MIB to be loaded is called IBMCLTM and is located in QSYS/QANMMIB file on the AS/400.

Figure 89 on page 140 shows the clientMgmtSubAgent MIB being selected for querying.

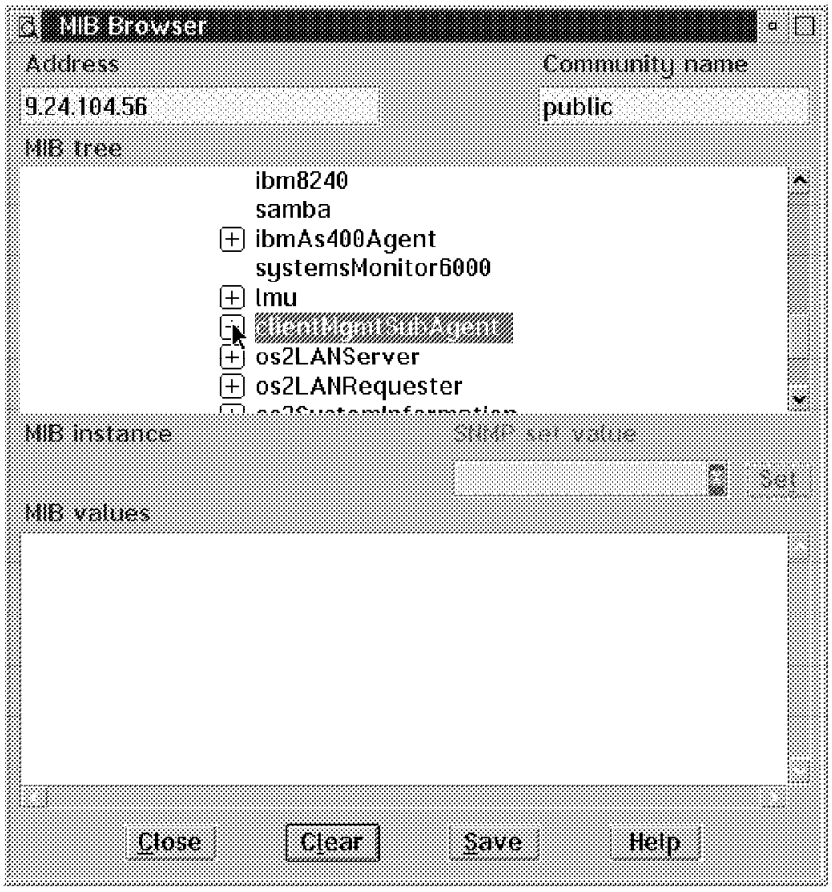


Figure 89. Selection of the Client Management MIB in NetView for OS/2

By clicking on the plus symbol (as shown in the picture) we are expanding the MIB tree.

clntSystem Figure 90 shows the clntSystem group being selected for querying.

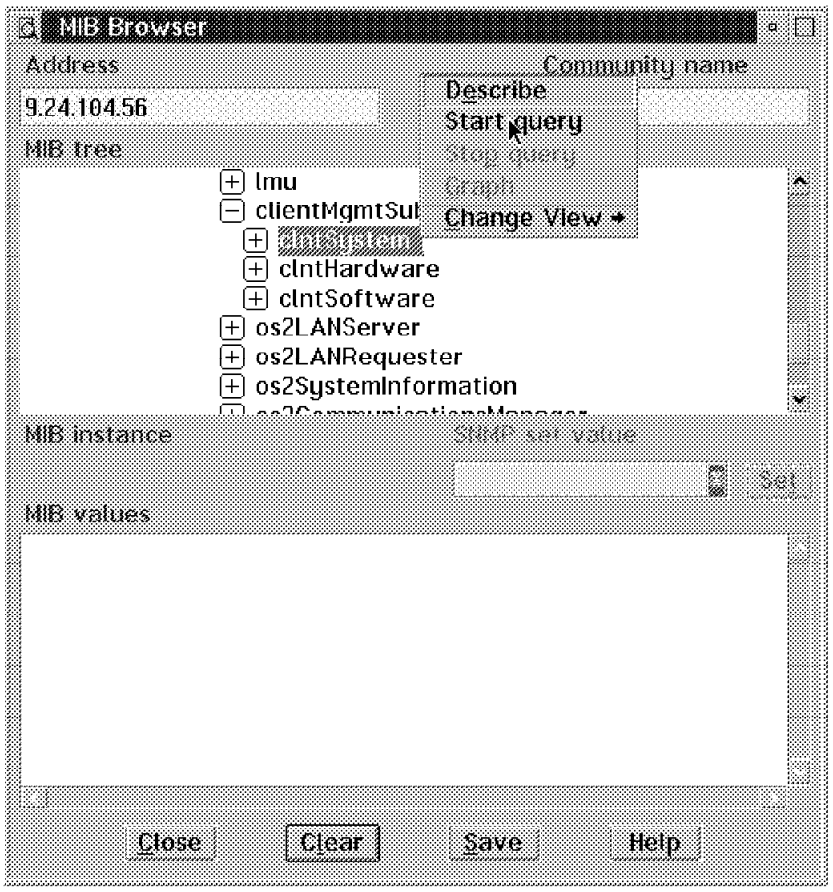


Figure 90. Selection of Query on the clntSystem Group

See Figure 91 on page 142 for the result of this query.

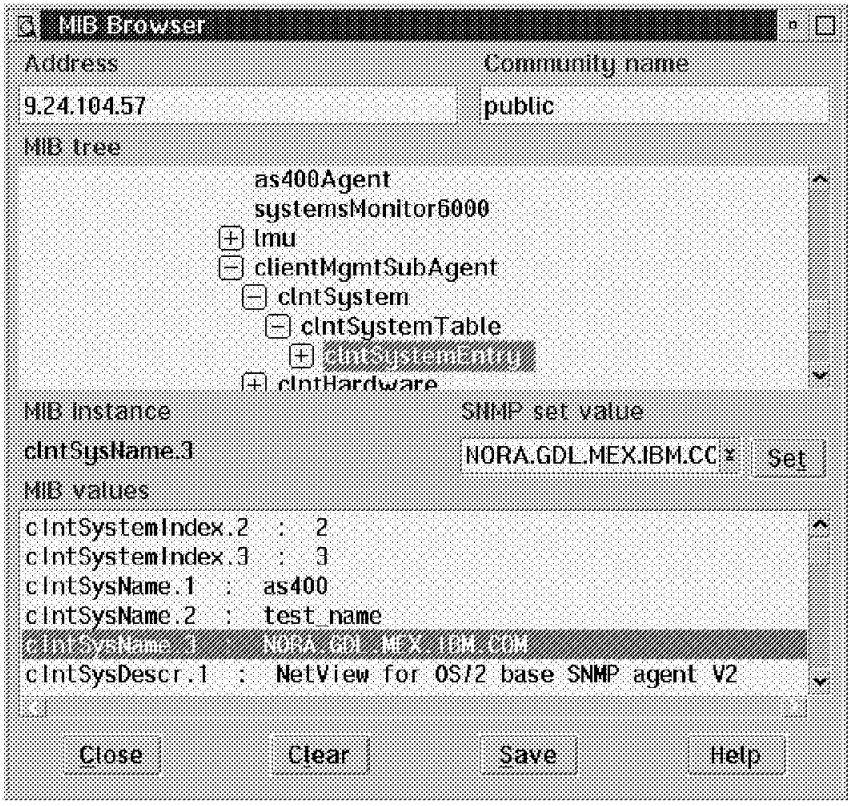


Figure 91. Results of the clntSystem Query

Figure 91 shows the results of the query performed on the *clntSystem* group. The information shown is from the clntSystemEntry MIB object within the clntSystemTable from the clntSystem group. Notice how the second line from the bottom in the MIB values area has been highlighted by placing the mouse pointer on that line and clicking on it. This puts information about the selected entry in the SNMP set value box of the screen in preparation for a SET operation.

clntHardware Figure 92 shows the results of the query performed on the clntHardware group.

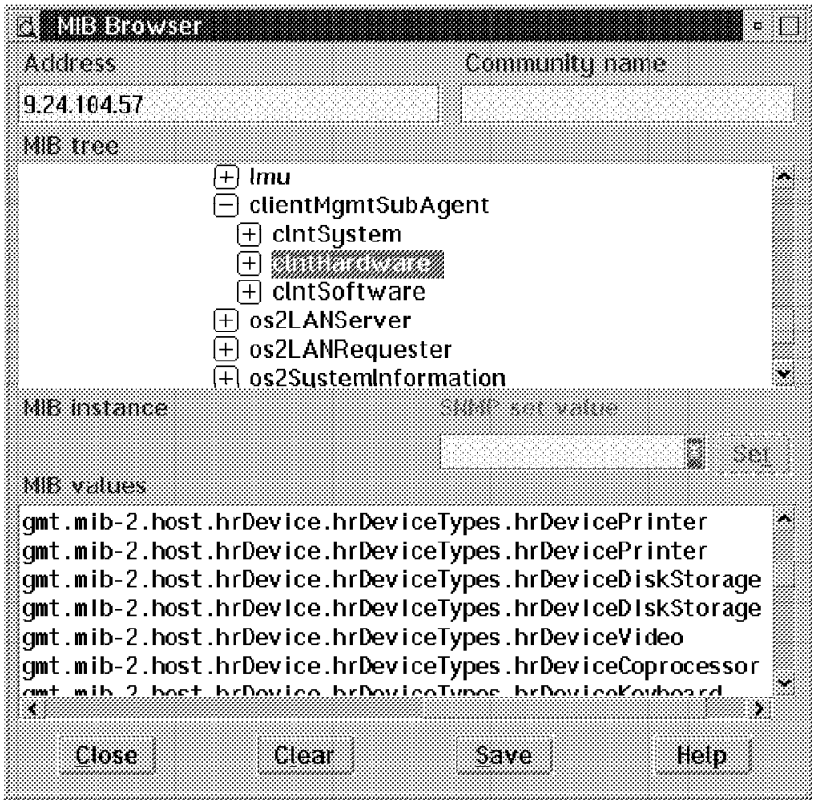


Figure 92. Results of Query on the clntHardware Group

In Figure 92 we can see part of the list of hardware devices from the client management database of system 9.24.104.57 (RALYAS4B). Note that if the Community name browser field is empty, the community name *public* will be used by default as specifically entered in Figure 91 on page 142.

Figure 93 shows the results of the query performed on the clntStorageDescr object. The clntStorageDescr MIB object is an entry (clntStorageEntry) within the clntStorageTable from the clntHardware group.

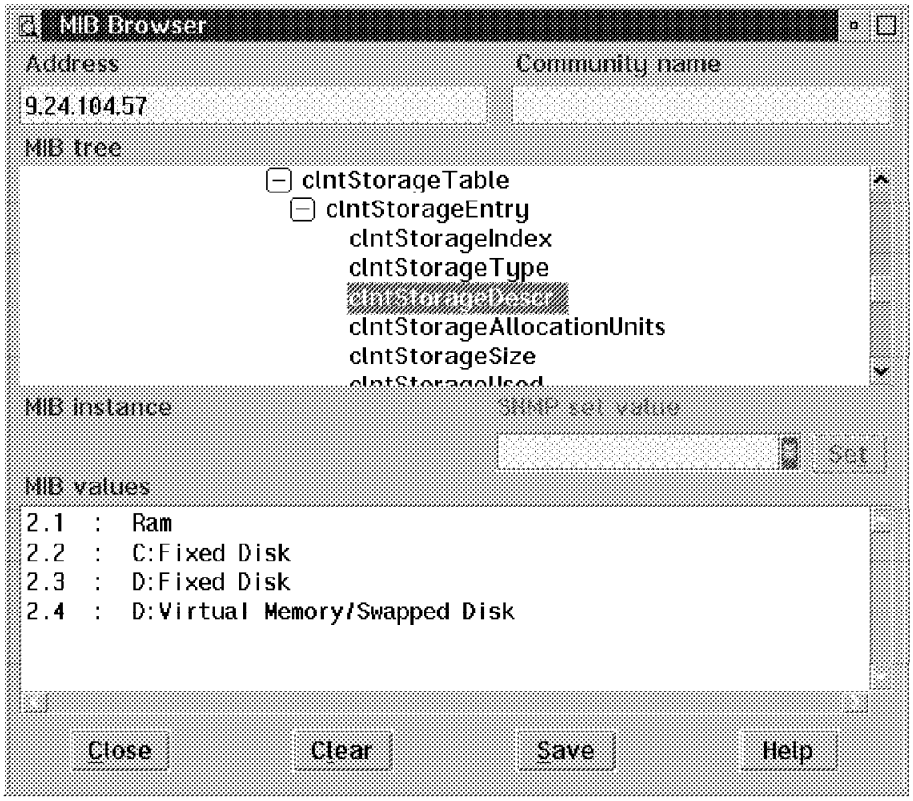


Figure 93. Results of Query on the clntStorageDescr Object

We can see from Figure 93 that this client system has the following storage devices:

- Random access memory
- Two OS/2 fixed disks, identified as C: and D:
- Part of fixed disk D allocated to a virtual memory/disk swap area

In Figure 94 on page 145 we can see the size of each of these storage areas.

Figure 94 shows the results of the query performed on the clntStorageSize object. The clntStorageSize MIB object is an entry (clntStorageEntry) within the clntStorageTable from the clntHardware group.

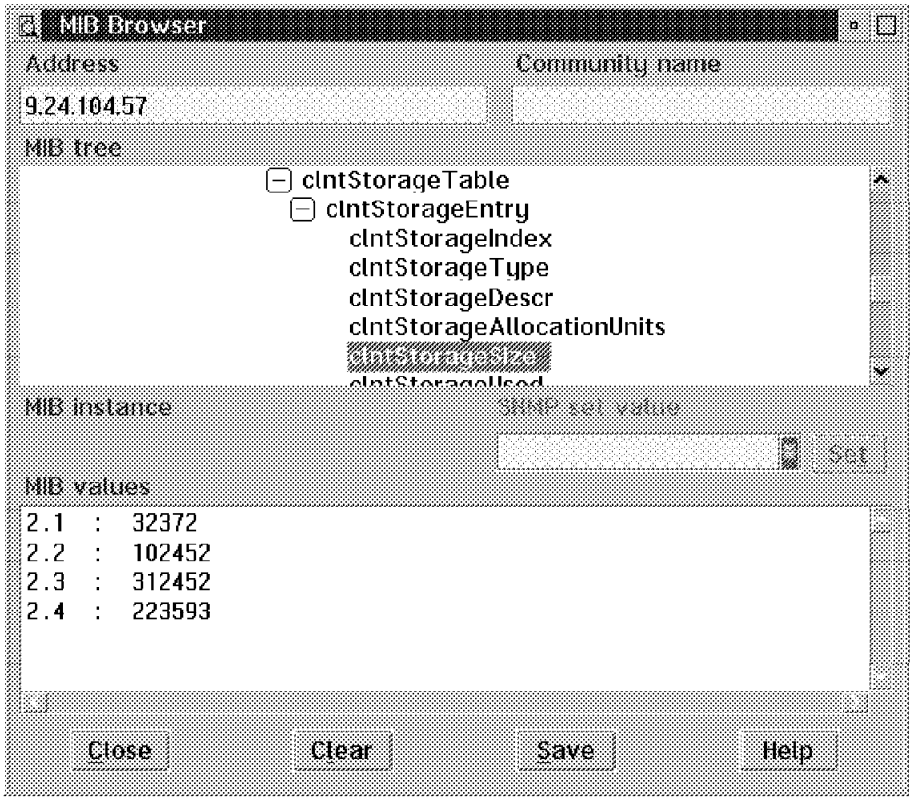


Figure 94. Results of Query on the clntStorageSize Object

We can see from Figure 94, for example, that the client system has 32 MB of random access memory installed and its C: disk has a size of 102 MB.

8.1.4 Client Access/400 Optimized for OS/2

In addition to being available via the AS/400 client inventory management application, the Client Access/400 Optimized for OS/2 SNMP MIBs can be queried directly from an SNMP manager. We cover this in 10.2, “Host Resources MIB and Client Access Optimized for OS/2” on page 197.

Chapter 9. OS/400 Supported MIBs

The following standard, IBM and Novell enterprise MIBs are supported by the AS/400:

Standard RFC MIBs

- MIB II (RFC1213)
- Host Resources MIB[1]
- Ethernet-like (RFC1398)
- FDDI (RFC1285)
- Frame Relay (RFC1315)
- Token-Ring (RFC1231)

DPI 2.0 (RFC1592)

SNMP subagent MIB

IBM enterprise MIBs

- APPN MIB
- Client management MIB
- NetView for AIX subagent MIB
- AS/400 Remote Workstation MIB[1]

Novell enterprise MIBs

- Internetwork Packet Exchange (IPX) MIB [2]
- Routing Information Protocol (RIP) MIB [2]
- NetWare Link Services Protocol (NLSP) MIB [2]

Notes:

[1]Not supported by V3R1 or V3R2

[2]Not supported by V3R1 or V3R6

Note

The token-ring and Ethernet MIB groups are supported, on the AS/400, only by the newer IOP hardware. That is the 2619 IOP for token-ring and the 2617 IOP for Ethernet. There is currently no SNMP support for the FSIOP (File Server IOP) or IPCS (Integrated PC Server).

For an explanation of the various MIB types see 2.9, “Understanding MIBs” on page 14.

For a summary of the supported MIBs in relation to the OS/400 version see the following table.

Table 3 (Page 1 of 2). The Supported MIB Groups

MIB Group	**Supported on Version**			
	V3R1[1]	**V3R2**	**V3R6**[1]	**V3R7**
MIB II	Yes	Yes	Yes	Yes
Host Resource	No	No	Yes	Yes
Ethernet-like	Yes	Yes	Yes	Yes

Table 3 (Page 2 of 2). The Supported MIB Groups

MIB Group	Supported on Version			
	V3R1[1]	**V3R2**	**V3R6**[1]	**V3R7**
FDDI	Yes	Yes	Yes	Yes
Frame Relay	Yes	Yes	Yes	Yes
Token-Ring	Yes	Yes	Yes	Yes
APPN	Yes	Yes	Yes	Yes
Client management	Yes	Yes	Yes	Yes
NetView for AIX subagent	Yes	Yes	Yes	Yes
DPI 2.0	Yes	Yes	Yes	Yes
SNMP subagent	Yes[2]	Yes	Yes	Yes
AS/400 Remote Workstation	No	No	Yes	Yes
Novell Enterprise MIBs	No	Yes	No	Yes

Notes:

[1]The installed version and appropriate PTFs.

[2]The V3R2 SNMP subagent MIB is treated as experimental-only.

9.1 Standard RFC MIBs

All devices that support SNMP are also required to support a standard set of common managed object definitions of which a MIB is composed. The standard MIB object definition, MIB-II, enables you to monitor and control SNMP managed devices.

9.1.1 MIB-II

MIB-II describes those objects that are implemented by managed nodes that run the Internet suite of protocols.

The AS/400 implements MIB-II. The group EGP (Exterior Gateway Protocol) is not provided by OS/400 SNMP because the AS/400 does not provide Exterior Gateway Support.

OS/400 supports the following MIB-II MIB groups:

System This group contains information about the agent system such as system name, system location and system contact.

Interfaces This group contains information about the interfaces on the agent system such as type of interfaces present, speed of the interfaces and a description of the interfaces.

AT (Address Translations) This group contains address translation information that provides an association between the network address and the physical address.

IP (Internet Protocol) This group contains Internet protocol information such as the number of datagrams sent and received by the agent system.

ICMP (Internet Control Message Protocol) This group contains ICMP information about the agent system such as the number of ICMP ECHO (PING) messages received.

TCP (Transmission Control Protocol) This group contains TCP information such as the number of TCP segments received and the number of TCP connections currently established.

UDP (User Datagram Protocol) This group contains UDP related data about the agent such as number of UDP datagrams delivered to UDP users.

Transmission (Transmission Media Specific) This group contains information about the various transmission protocols used by the agent system such as token-ring statistics and Ethernet statistics.

SNMP (SNMP Application entities) This group contains SNMP related data such as the number of requests received with an unknown community name.

Note

To retrieve data from the transmission group you must make sure that the relevant IOP is active. To do this, check the MIB variable IfOperStatus under the interfaces group of MIB-II. The IOP should be in UP status.

RFC: RFC1213

RFC Noted Exceptions: The egp group is not supported. The set operation is not supported for the following MIB object (which is defined as having read-write access):

ifAdminStatus

Note that ifAdminStatus tracks the desired state of a network interface as set with an OS/400 command (for example, NETSTAT). Therefore, the get operation can still be used on ifAdminStatus and ifOperStatus to determine if there is a problem with an interface.

The set operation is accepted for the following MIB objects, which are defined as having read-write access. However, the values will not change as a result of the set operation. This behavior allows an SNMP manager to successfully perform a set operation on an entire row of a table. A subsequent get operation will show which values have actually changed.

- ipRouteIfIndex
- ipRouteMetric1
- ipRouteMetric2
- ipRouteMetric3
- ipRouteMetric4
- ipRouteAge
- ipRouteMetric5

For some MIB objects, the set operation is not supported for all values that are defined as being valid. These MIB objects are listed here, along with the values to which they can be set.

ipRouteType	invalid(2), indirect(4)
ipNetToMediaType	invalid(2), static(4)
tcpConnState	deleteTCB(12)

In order to set the value of atNetAddress, its syntax must be encoded as OCTET STRING, rather than NetworkAddress. Note that it is never necessary to set the

value of atNetAddress directly. This is because a row can be added to or deleted from the atTable by setting the value of only atPhysAddress.

The result of changing the value of a MIB object that is an index to an instance of that MIB object, is undefined. For example, the result of this operation is undefined:

```
set ipRouteDest.9.130.38.28=9.130.38.29.
```

When adding a row to a table by setting the value of a MIB object, default values are assigned to the other objects in the row.

- *indexes*: The value of a MIB object which is an index to an instance of that MIB object need not be set explicitly. For example, the operation:

  ```
  set ipRouteNextHop.9.130.38.28=9.130.25.250
  ```

  will implicitly create the MIB object instance ipRouteDest.9.130.38.28, with the value 9.130.38.28.

- *atPhysAddress*: There is no default value for this MIB object. The value of this MIB object must be set in order to create a new row in the atTable.
- *ipRouteNextHop*: There is no default value for this MIB object. The value of this MIB object must be set in order to create a new row in the ipRouteTable.
- *ipRouteType*: indirect(4).
- *ipRouteMask*: Corresponds to the network class. For example, the mask of a class-A network will default to 255.0.0.0.
- *ipNetToMediaPhysAddress*: There is no default value for this MIB object. The value of this MIB object must be set in order to create a new row in the ipNetToMediaTable.
- *ipNetToMediaType*: static(4).

MIB Subtree Description: This is the management information base for network management of TCP/IP-based internets.

MIB Subtree Object Identifier: mib-2 ::= { mgmt (1) }

Prerequisite MIB Modules: RFC1155, RFC1212

9.1.1.1 Browsing the MIB-II MIB

For an example of browsing MIB-2 objects, see 5.1.4, “Using the MIB Browser” on page 50.

9.1.2 Host (Host Resources) MIB

This group contains host-related data such as the hardware and software installed. We look at this MIB in greater detail in Chapter 10, “Host Resources MIB” on page 189.

9.1.3 Ethernet-Like Interface MIB

The Ethernet-like Interface MIB defines objects for managing Ethernet-like object interface types.

The Ethernet-like MIB is contained in the Transmission group of MIB-II.

OS/400 provides support for the following Ethernet-like MIB groups and tables:

- dot3StatsTable (Statistics group)
- dot3Tests (Tests table)
- dot3ChipSets (Hardware Chipsets table)

RFC: RFC1398

RFC Noted Exception: The dot3CollTable is not supported.

MIB Subtree Description: Ethernet-like MIB.

MIB Subtree Object Identifier: dot3 ::= { transmission 7 }

Prerequisite MIB Modules: RFC1155, RFC1213, RFC1212

9.1.4 Browsing the Transmission Group MIB

For the objects under the Transmission group, Ethernet, FDDI, frame relay and token-ring we will use the NetView for OS/2 MIB Browser to look at some example data. As described earlier, the status of the interface must be up. The first thing we will do is check the status of the interface. Select the following MIB objects until you get to the ifEntry screen as shown in Figure 95 on page 152.

mgmt→
mib-2→
interfaces→
ifTable→
ifEntry

See 5.1.4, “Using the MIB Browser” on page 50 for an example of how to use the Browser.

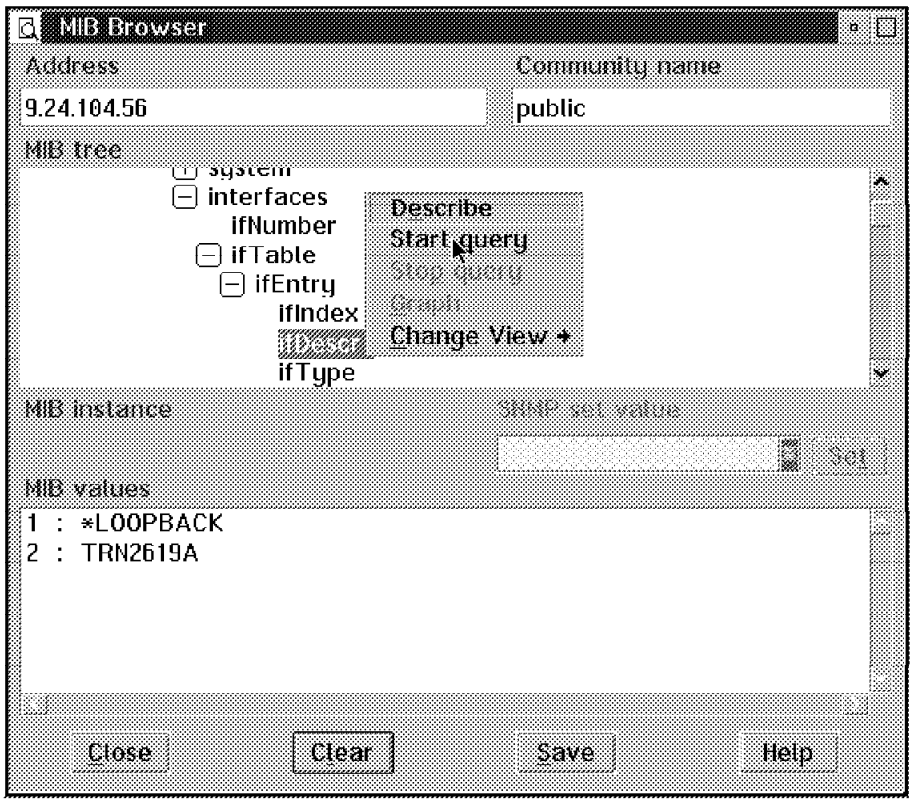
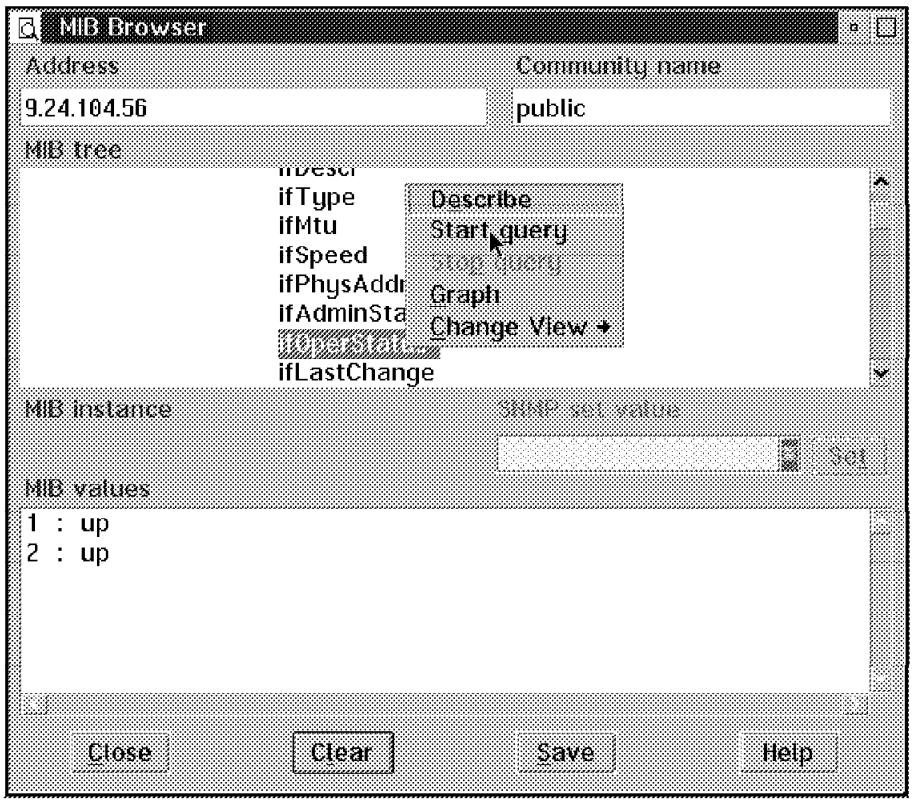


Figure 95. Working with Interfaces

Now click on **ifDesc** and click on the **Start query** button. This should present you with the interfaces on the system. Next to the interface will be an index number. In our case the line TRN2619A has an index number of 2. For subsequent queries of interface data the index will remain constant and any data retrieved with an index number of 2 will be associated with TRN2619A.

You can scroll down the MIB tree and perform a query on the MIB object ifOperStatus. As you can see in Figure 96 on page 153 the index number 2 is in UP status, meaning that TRN2619A is up, as is the other interface.



Status

Figure 96. Result of Query of ifOperState

Having verified that the interfaces are up, we can run queries on the objects within the transmission group. The group is accessed by selecting the following objects:

mgmt→
mib-2→
transmission

9.1.5 FDDI MIB

The FDDI MIB defines objects for managing devices which implement FDDI.

The FDDI MIB is contained in the Transmission group of MIB-II.

OS/400 provides support for the following FDDI MIB groups:

- snmpFddiSMT (SMT group)
- snmpFddiMAC (MAC group)
- snmpFddiPATH (PATH group)
- snmpFddiPORT (PORT group)
- snmpFddiChipSets (Chip Set group)

RFC: RFC1285

RFC Noted Exceptions: The Attachment group is not supported. The set operation is not supported for this MIB.

MIB Subtree Description: FDDI MIB.

MIB Subtree Object Identifier: fddi ::= { transmission 15 }

Prerequisite MIB Modules: RFC1155, RFC1213, RFC1212

9.1.5.1 Browsing the FDDI MIB

From the transmission group, select the **fddi** MIB object.

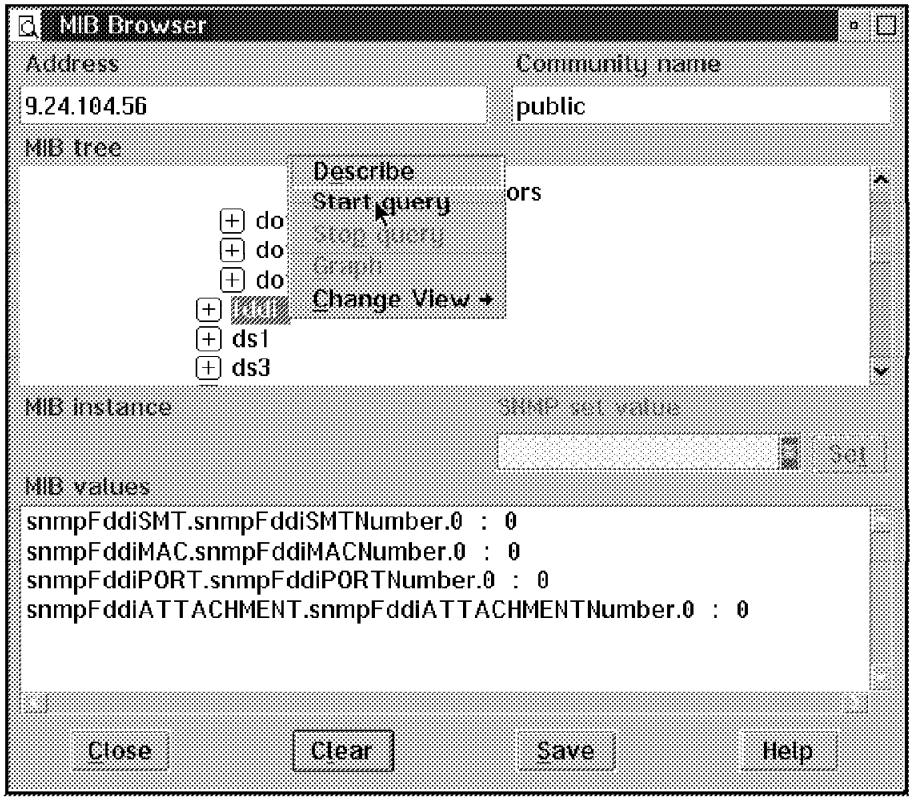


Figure 97. Browsing the FDDI MIB

Figure 97 shows some of the data that can be retrieved using the FDDI MIB.

9.1.6 Frame Relay MIB

The frame relay MIB defines objects for managing frame relay interfaces.

OS/400 provides support for the following frame relay MIB tables:

- frDlcmiTable (Data Link Connection Management Interface Table)
- frCircuitTable (Circuit Table)
- frErrTable (Error Table)

RFC: RFC1315

RFC Noted Exception: The set operation is supported for only frTrapState.

MIB Subtree Description: Frame Relay MIB.

MIB Subtree Object Identifier: frame-relay ::= { transmission 32 }

Prerequisite MIB Modules: RFC1155, RFC1213, RFC1212, RFC1215

9.1.6.1 Browsing the Frame Relay IMIB

From the transmission group, click on the **frame-relay** MIB object.

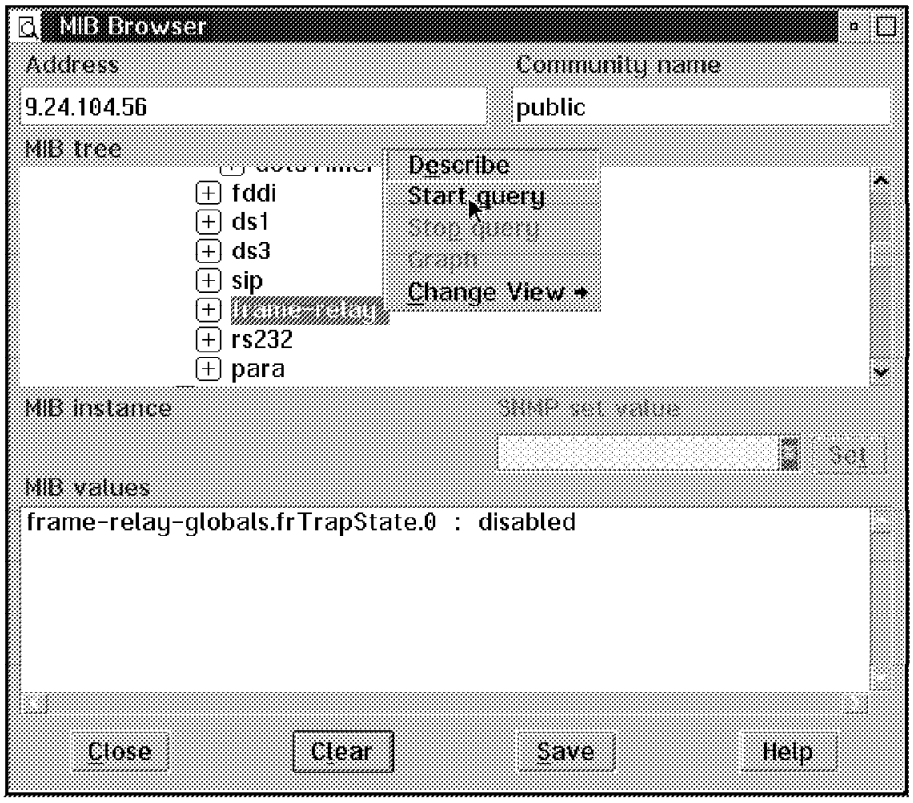


Figure 98. Browsing the Frame Relay MIB

Figure 98 shows some of the data that can be retrieved from the frame relay MIB.

9.1.7 IEEE 802.5 Token-Ring MIB

The IEEE 802.5 token-ring MIB defines objects for managing subnetworks which use the IEEE 802.5 token-ring technology described in 802.5 token-ring Access Method and Physical Layer Specifications, IEEE Standard 802.5-1989.

The token-ring MIB is contained in the Transmission group of MIB-II.

OS/400 provides support for the following token-ring MIB tables:

- dot5Table (Interface table)
- dot5StatsTable (Statistics table)

RFC: RFC1231

RFC Noted Exception: The dot5TimerTable and the set operation are not supported for this MIB.

MIB Subtree Description: Token Ring MIB

MIB Subtree Object Identifier: dot5 ::= { transmission 9 }

Prerequisite MIB Modules: RFC1155, RFC1212

9.1.7.1 Browsing the Token-Ring MIB

Using the MIB Browser it is possible to retrieve a variety of information from this MIB group. From the transmission object select the **dot5** object. See Figure 99 and Figure 100 on page 157 for an example of the data that can be retrieved from the token-ring MIB subtree.

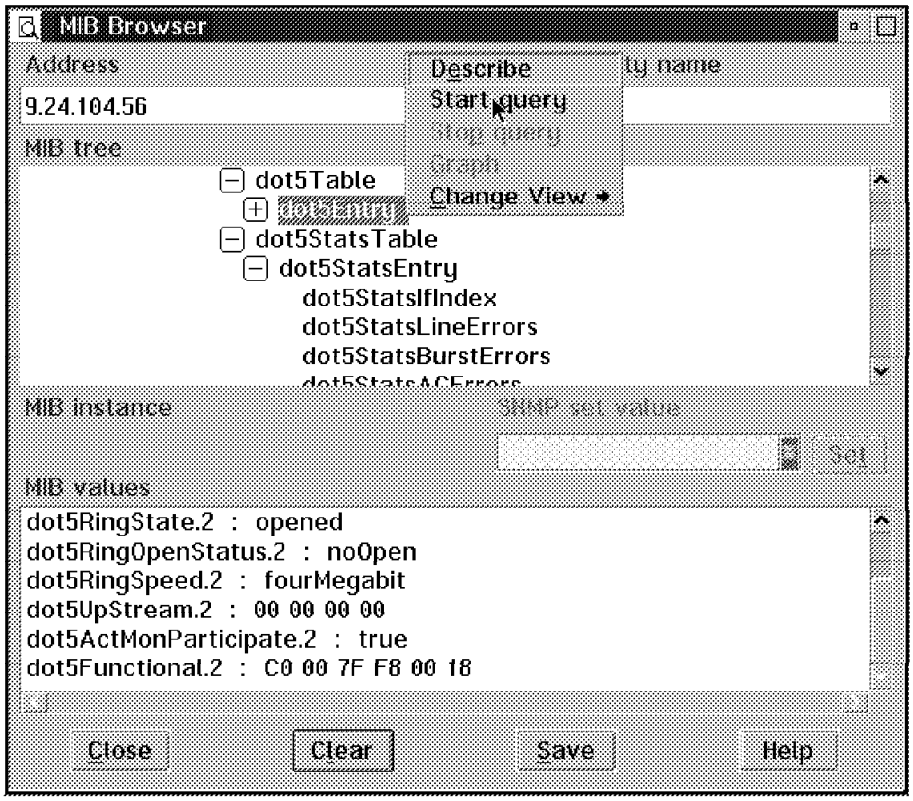


Figure 99. Token-Ring Specific Information

In Figure 99 you can see that we retrieved information from the dot5Entry group. The MIB value keywords, such as dot5RingSpeed, are all suffixed by a number (in this case 2). This is the index that is used to associate the value with a specific interface. See Figure 95 on page 152 for the interface names. In this case you can see that TRN2619A runs at four megabits (4 Mbps).

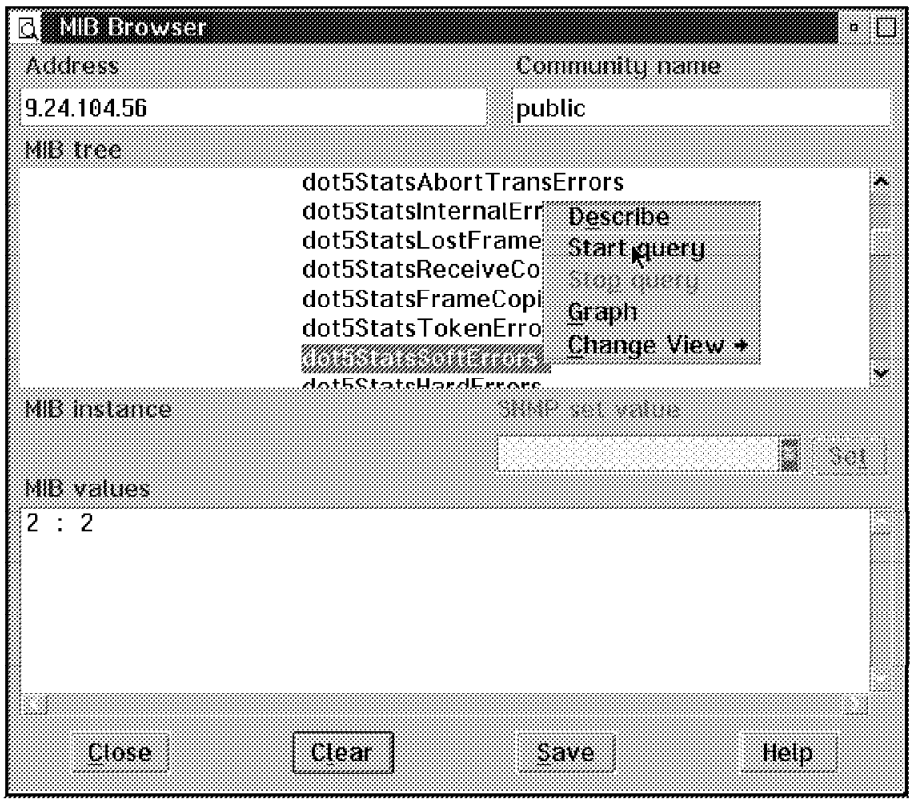


Figure 100. Statistical Information about the Token-Ring

In Figure 100 we can see that TRN2619A has logged two soft errors.

Remember that you can use the Describe button to obtain a description of the variable you selected.

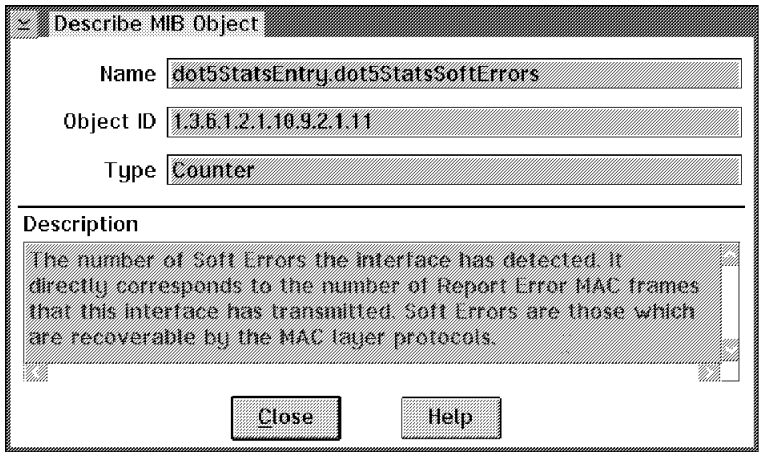


Figure 101. Description of the Variable

9.2 DPI 2.0 MIB

The DPI 2.0 MIB is an extension to SNMP agents that permits end-users to dynamically add or replace management variables in the local MIB without requiring recompilation of the SNMP agent. Although SNMP subagents are supported by an AS/400 system, this support does not use the DPI 2.0 MIB. This is because the AS/400 subagent support does not use TCP or UDP between the agent and subagents. A transparent, internal mechanism is used by the subagents. This MIB is supported by the AS/400 SNMP agent even though the communication between the agent and subagent on the AS/400 does not use TCP or UDP. This is done for compatibility.

This MIB is implemented by the AS/400 system but is not currently used. For further details on the DPI 2.0 MIB, see RFC1592.

RFC: RFC1592

RFC Noted Exception: None.

MIB Subtree Description: SNMP Distributed Protocol Interface Version 2.0

MIB Subtree Object Identifier: dpi20MIB ::= { ibmDPI 1 }

Prerequisite MIB Modules: SNMPv2-SMI

9.3 SNMP Subagent MIB

The SNMP subagent MIB provides information about subagents to facilitate management activities. The MIB is designed to handle multiple types of subagents, including DPI subagents (see RFC1592), for which the DPI 2.0 API is provided. All the SA MIB information is dynamic for the duration of the SNMP agent job. The SA MIB consists of six summary OIDs and two tables. The summary OIDs are as follows:

saDefaultTimeout
This is the default value for subagents for response timeout.

saMaxTimeout
This is the largest value a subagent may use for response time-out.

saAllowDuplicateIDs
This is the flag to allow duplicate subagent OIDs, or not.

saNumber
This is the number of currently registered subagents.

saAllPacketsIn
This is the total number of subagent packets that are received from all subagents.

saAllPacketsOut
This is the total number of subagent packets that are sent to all subagents.

The saTable provides information about specific subagents such as status, address, and description.

The saTreeTable provides information about specific subtrees that subagents have registered with the SNMP agent.

RFC: None

RFC Noted Exception: None

MIB Subtree Description: SNMP subagent MIB

MIB Subtree Object Identifier: saMIB ::= { internet(1) private(4) enterprise(1) ibm(2) ibmResearch(2) 12 }

MIB Module Name: IBMSNMPSA

Prerequisite MIB Modules: RFC1155, RFC1212, RFC1213

9.4 IBM Enterprise MIBs

SNMP permits vendors to define MIB extensions or enterprise-specific MIBs, specifically for controlling their products. An enterprise-specific MIB must follow certain definition standards just as other MIBs must, to ensure that the information they contain can be accessed and modified by SNMP agents. Some of the IBM enterprise-specific MIBs are discussed below.

Before a browser can access an enterprise MIB, the MIB descripition must be loaded into the browser. Enterprise MIB descriptions are distributed through the following mechanisms:

- If you have Internet access with FTP capability, MIB modules can be obtained from the NetView Association MIB server at `netview.cary.ibm.com`.
- IBM enterprise MIB modules supported by OS/400 are shipped with OS/400. Each MIB module is contained in a member of physical file QANMMIB in library QSYS. The physical file member name for each MIB module is listed as the MIB module name in the section that describes each enterprise MIB in this chapter.
- If you have Internet access with FTP capability, MIB modules for vendor MIBs registered under the enterprises subtree (including IBM MIBs) can be obtained by anonymous FTP from the Internet Assigned Numbers Authority (IANA) anonymous FTP server at `venera.isi.edu`. For example use a Web browser to: `FTP://venera.isi.edu/MIB`. For who/what is IANA see B.1.1, “Request for Comments (RFC)” on page 246.

9.4.1 Advanced Peer-To-Peer Networking (APPN) MIB

The APPN MIB subtree has three children which form the following groups:

ibmappnNode The APPN Node Group provides global information about the APPN node, which is either a network node or an end node.

ibmappnNnTopology The APPN Topology Group represents the entire APPN network topology including network nodes, virtual nodes, and all transmission groups (TGs) that are associated with these nodes.

ibmappnLocalTopology The APPN Local Topology Group describes the local topology. This MIB group defines the required objects for retrieval of information about this node and the objects that represent the local topology about end nodes.

The OS/400 V3R6 SNMP enhancements include the ability to use SNMP to change the status (vary off/vary on) of APPC lines and controllers using the set operation. We will discuss this in detail in 9.4.1.2, “Using the APPN MIB set function to Manage AS/400 Lines and Controllers” on page 167.

RFC: Informational RFC1593

RFC Noted Exceptions: Some groups of RFC1593 are implemented with some extensions. Refer to the concise MIB description (MIB module) for details. Browsing the physical file member IBMAPPN in file QSYS/QANMMIB on the AS/400 provides a useful explanation of this MIB’s function.

MIB Subtree Description: SNA APPN MIB

MIB Subtree Object Identifier: ibmappn ::= { internet(1) private(4) enterprises(1) ibm(2) prod(6) ibm6611(2) 13 }

MIB Module Name: IBMAPPN

Prerequisite MIB Modules: RFC1155, RFC1212, IBMMIB

APPN MIB Object Groups Supported: The following groups are supported under the ibmappnNode subtree:

ibmappnGeneralInfoAndCaps
ibmappnNnUniqueInfoAndCaps
ibmappnEnUniqueInfoAndCaps
ibmappnPortInformation
ibmappnLinkStationInformation
ibmappnSnmpInformation

Note

The ibmappnPortInformation and ibmappnLinkStationInformation MIB objects are not supported by V3R1 or V3R2.

The following groups are supported under the ibmappnNn subtree:

ibmappnNnTopo (The following 4 objects are supported):

ibmappnNnTopoMaxNodes
ibmappnNnTopoCurNumNodes
ibmappnNnTopoNodePurges
ibmappnNnTopoTgPurges

ibmappnNnTopology - tables for APPN network topology (the following groups/tables are supported):

ibmappnNnTopologyTable
ibmappnNnTgTopologyTable
ibmappnNnTopologyFRTable
ibmappnNnTgTopologyFRTable

The following groups are supported under the ibmappnLocalTopology subtree:

ibmappnLocalThisNode (the following groups/tables are supported):

ibmappnLocalGeneral
ibmappnLocalNnSpecific

ibmappnLocalTg - table of local transmission groups (TGs)

ibmappnLocalEnTopology (the following tables are supported):

ibmappnLocalEnTable - table of adjacent end nodes
ibmappnLocalEnTgTable - table of adjacent end node TGs

9.4.1.1 Browsing the APPN MIB

Having looked at which groups are supported under the ibmappn MIB, let us use the NetView for OS/2 MIB Browser to retrieve some data from an AS/400 using this MIB.

Before we can query the ibmappn MIB objects, we must load this enterprise MIB into NetView for OS/2. To do this we follow the steps shown in Chapter 13, “Loading an Enterprise-Specific MIB” on page 219. The MIB to be loaded is called IBMAPPN and is located in QSYS/QANMMIB.

Note

If the APPN MIB had been used prior to V3R6 then the MIB must be re-loaded into the SNMP manager before the APPN MIB enhancements can be used. The NetView for OS/2 commands to unload then re-load the MIB are as follows:

```
LOADMIB -UNLOAD APPNMIB
LOADMIB -LOAD APPNMIB
```

See Chapter 13, “Loading an Enterprise-Specific MIB” on page 219 for detailed information about loading a MIB file.

Use the MIB Browser to get to the ibmappn MIB group. See 5.1.4, “Using the MIB Browser” on page 50 for an explanation on how to use the MIB Browser. To get to the ibmappn MIB you will need to follow this path:

private→
enterprises→
ibm→
ibmProd→
ibm6611→
ibmappn

This will be our starting point, on the MIB tree, for the following examples.

ibmappnNode: First let's look at the ibmappnNode subtree. From ibmappn select **ibmappnNode**.

We started a query on the group ibmappnGeneralInfoAndCaps, which gave us lots of information about the system we queried, RALYAS4B. You can see, in Figure 102, information such as the CP name of the node and the time that the node has been running since the last IPL.

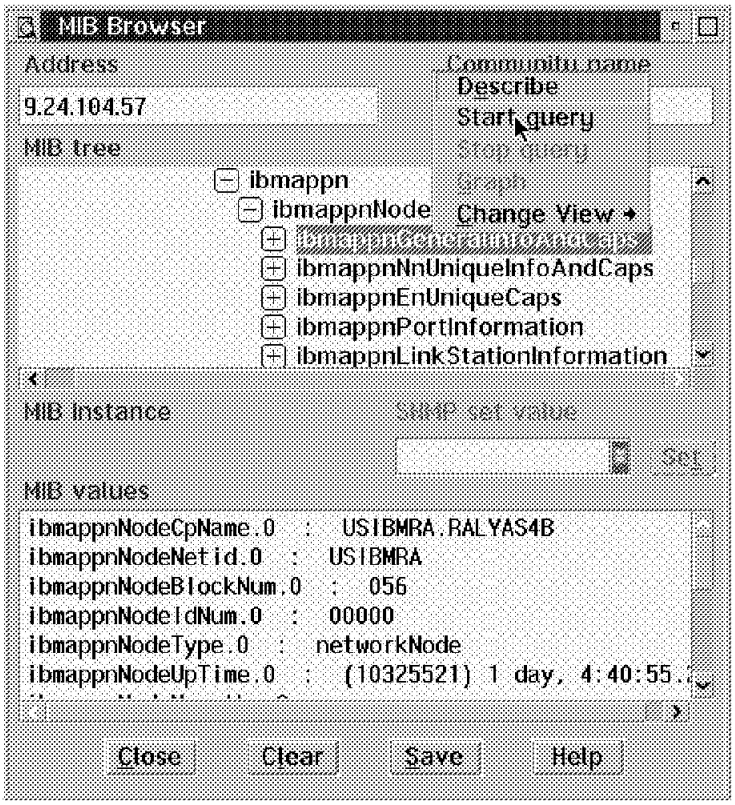


Figure 102. Browsing the ibmappnGeneralInfoAndCaps MIB Group

ibmappnNnToplogy: Next we look at the ibmappnNn subtree. From ibmappn, take the following path:

ibmappnNn→
ibmappnNnTopology→
ibmappnTopologyTable→
ibmappnTopologyEntry

Figure 103 shows the results of querying the ibmappnNodeName variable. There are many instances of this variable on the AS/400: one instance for every APPN node stored in the APPN topology database.

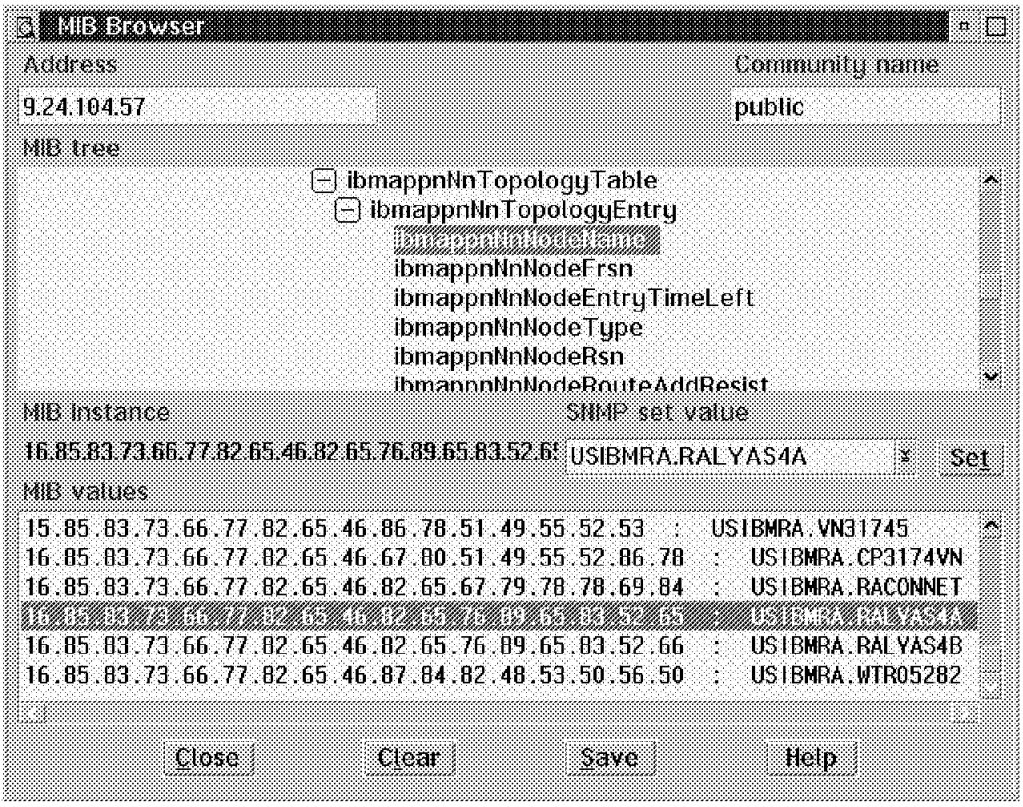


Figure 103. Retrieving the Node Names from the APPN Topology Database

The data returned is a string of ASCII data preceded by a length indicator; in the example shown, the length is always 16 bytes. The ASCII data is an index as an identifier of an instance. NetView for OS/2 has translated the data into text. When we look at a different variable in the group, such as the ibmappnNnNodeQuiescing variable, the returned value is merely an integer 1 or 2. This is translated by NetView for OS/2 into a yes or no value. To determine which instance of this value is associated with the corresponding node, the index value must be used.

In Figure 104 on page 164 you can see that a returned value of no has been highlighted. The ASCII data to the left is the identifier of the specific instance. If you look at the highlighted data in Figure 103 you can see that the identifier is the same; therefore, we can say that the two values are for the same instance and that the APPN node USIBMRA.RALYAS4A is not quiescing.

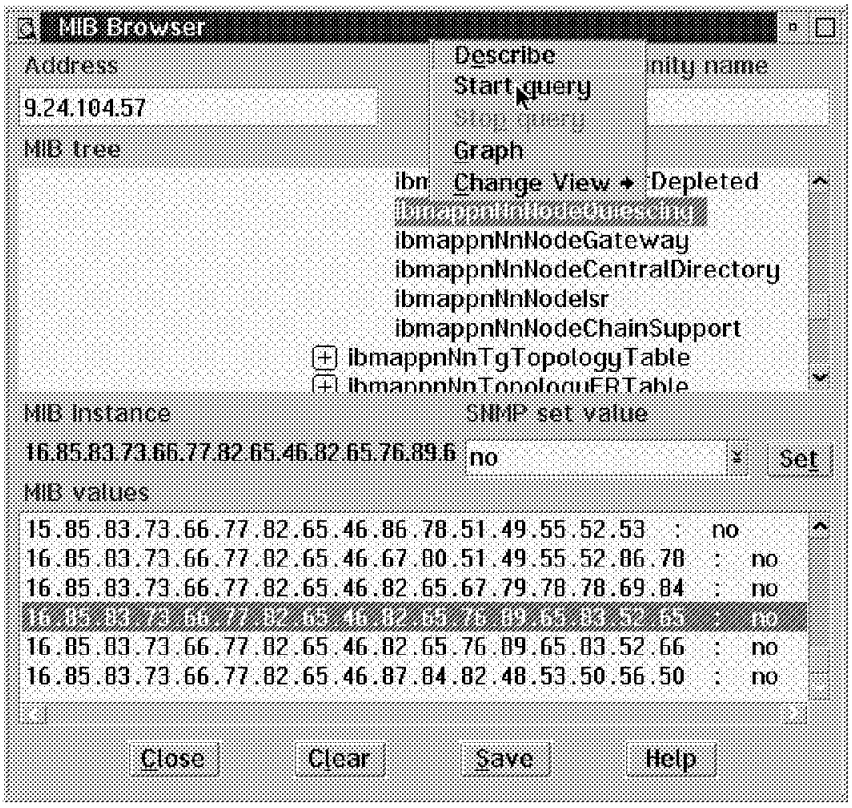


Figure 104. Is the APPN Node Quiescing?

Don't forget that you can click on the **describe** button for a description of the variable. See Figure 105.

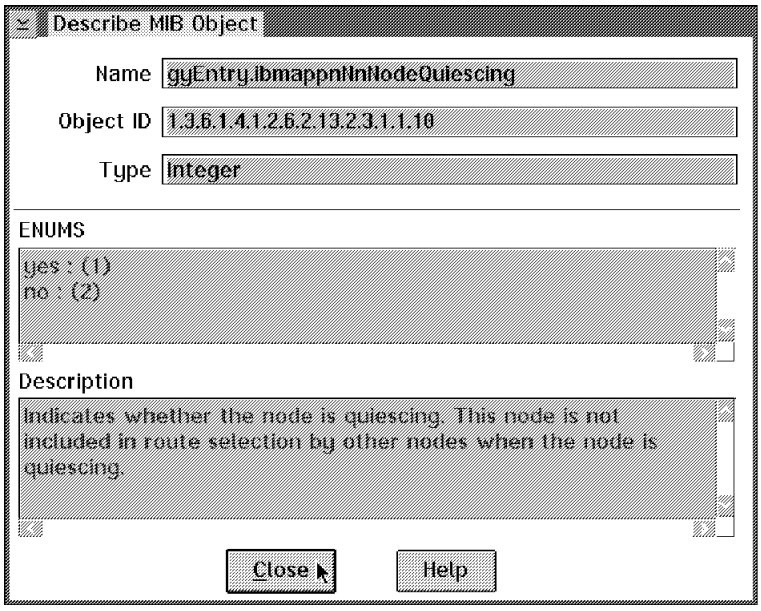


Figure 105. Description of the ibmappnNnNodeQuiescing Variable

Figure 106 shows the results of querying the variable ibmappnNodeEntryTimeLeft. This is the number of days left before the system is purged from the topology database.

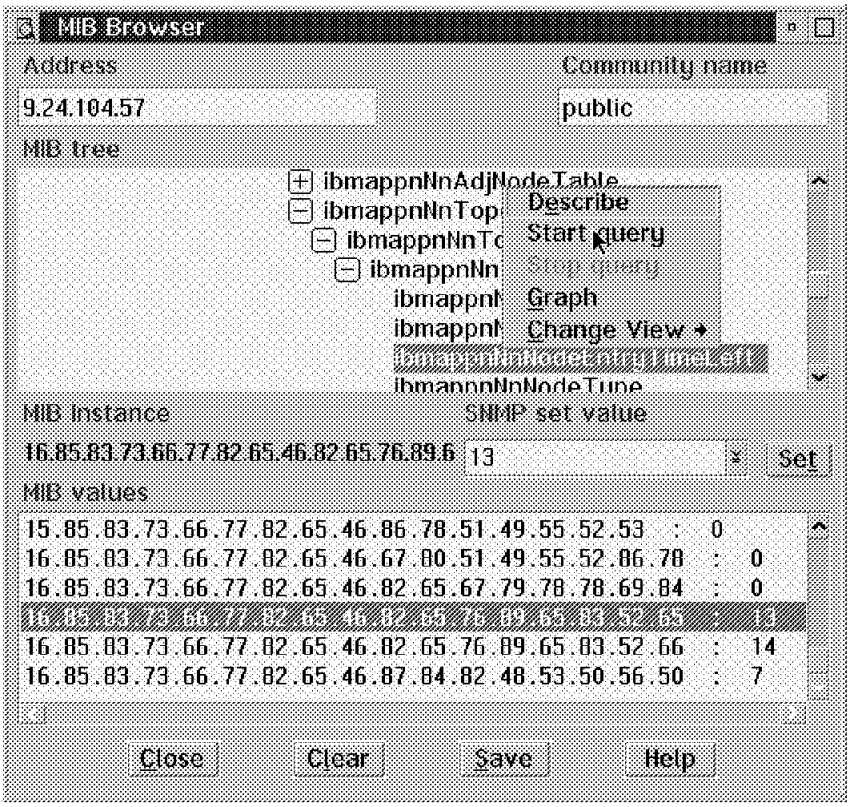


Figure 106. Time Left in Database before Deletion

ibmappnLocalToplogy: Now we will take a look down the ibmappnLocalTopology subtree. Using the MIB Browser, select the following objects until you get to the ibmappnLocalTgEntry group:

ibmappnLocalTopology→
ibmappnLocalThisNode→
ibmappnLocalTg→
ibmappnLocalTgTable→
ibmappnLocalTgEntry

We queried the variable ibmappnLocalTgDest which lists the destination nodes available over this node's transmission groups. Figure 107 shows some of the destination systems available from RALYAS4B.

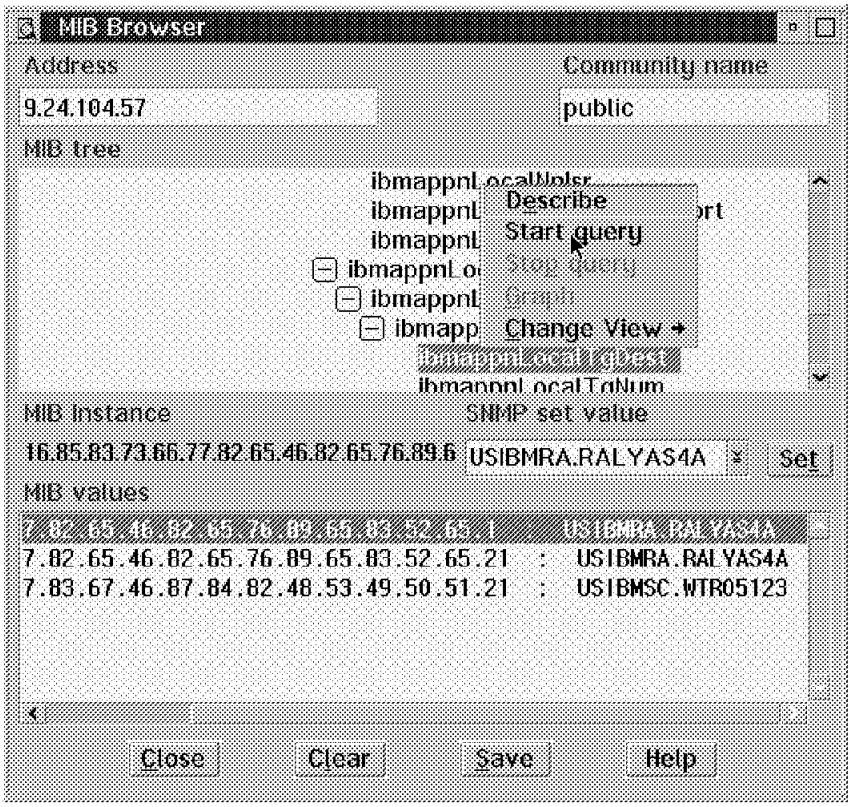


Figure 107. Destination Nodes of Transmission Groups

Once again the ASCII data on the left is used as an identifier of a specific instance. With this in mind look at Figure 108 on page 167. This shows the variable ibmappnLocalTgOperational and indicates whether the transmission group is operational or not.

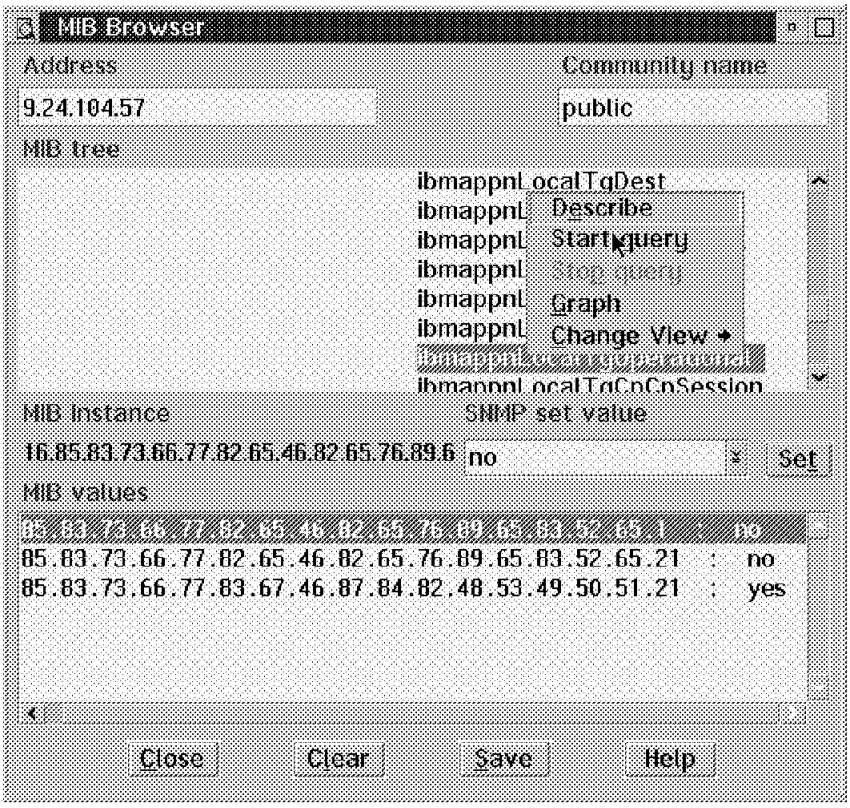


Figure 108. Is the Transmission Group Operational?

You can see that a transmission group to USIBMRA.RALYAS4A is not operational.

The examples above showed just a few of the many variables available via the APPN MIB.

Note

Although we did not have time to have a look at it during the residency that resulted in this book, Router and Bridge Manager/6000 is able to display the APPN topology in a more user-friendly way using this same APPN MIB. A graphical representation is given.

9.4.1.2 Using the APPN MIB set function to Manage AS/400 Lines and Controllers

The OS/400 V3R6 SNMP enhancements include improved APPN MIB support. The SNMP set function can now be used to change the status of APPC lines and controllers via two variables in the ibmappnNode subtree. The ibmappnNodePortState MIB variable can be used to display and, if necessary change, the status of APPN capable line descriptions. The ibmappnNodeLsState MIB variable can be used to display and, if neccessary change, the status of APPN capable controller description. As an example, let's look at how we can use the ibmappnNodeLsState MIB variable to activate (vary on) an inactive (varied off) controller description. In Figure 109 on page 168 we can see that the controller state is currently varied off.

```
                        Work with Configuration Status                RALYAS4C
                                                                 08/08/96  13:26:57
Position to  . . . . .               Starting characters

Type options, press Enter.
  1=Vary on   2=Vary off   5=Work with job   8=Work with description
  9=Display mode status ...

Opt  Description       Status               -------------Job--------------
     RALYAS4B          VARIED OFF
       RALYAS4B        VARIED OFF









                                                                        Bottom
Parameters or command
===>
F3=Exit   F4=Prompt   F12=Cancel   F23=More options   F24=More keys
```

Figure 109. The Status before Setting the ibmappnNodeLsState Variable

The ibmappnNodeLsState MIB variable is located within ibmappnNodeLsTable. To locate the ibmappnNodeLsTable table, follow this path:

private→
enterprises→
ibm→
ibmProd→
ibm6611→
ibmappn→
ibmappnNode→
ibmappnLinkStationInformation→
ibmappnNodeLsTable

Managing Line Descriptions

The ibmappnNodePortState MIB variable is located within the ibmappnNodePortTable.

To display/change the status of a line description follow this path:

private→
enterprises→
ibm→
ibmProd→
ibm6611→
ibmappn→
ibmappnNode→
ibmappnPortInformation→
ibmappnNodePortTable

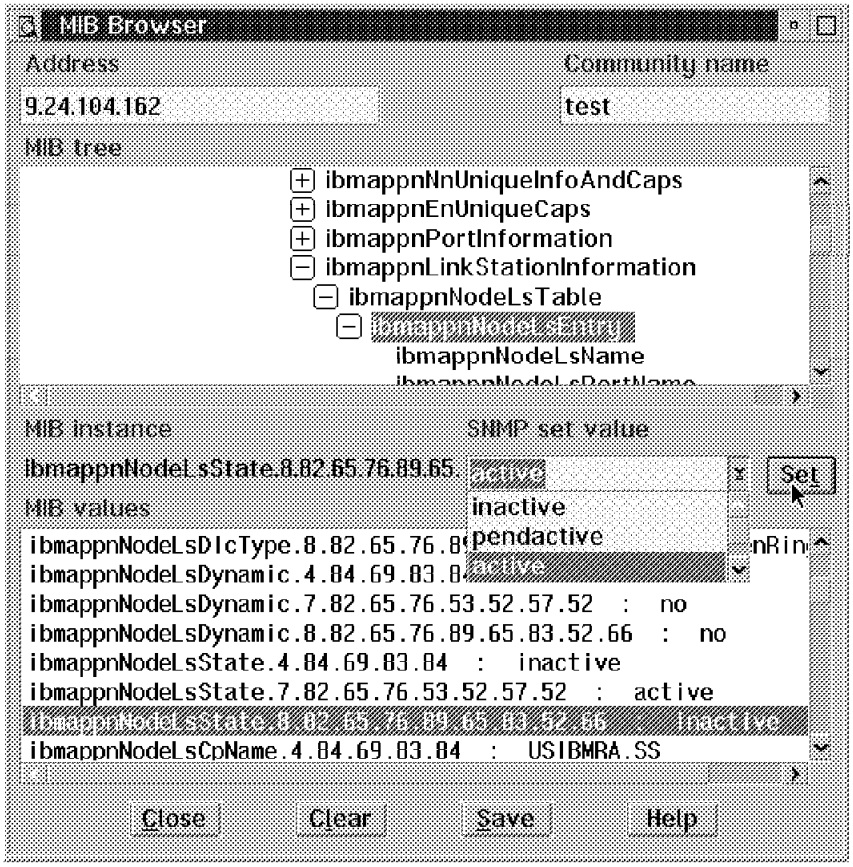


Figure 110. Setting the ibmappnNodeLsState MIB Variable

Having selected the ibmappnNodeLsState MIB object for the appropriate controller description via the unique instance value for that controller description (as determined via the ibmappnNodeLsName MIB object), we select a set value of active from the SNMP set value window, then press the Set button. Note that SNMP community *test* allows write access. If the set operation is successful you will see the completion message, as shown in Figure 111.

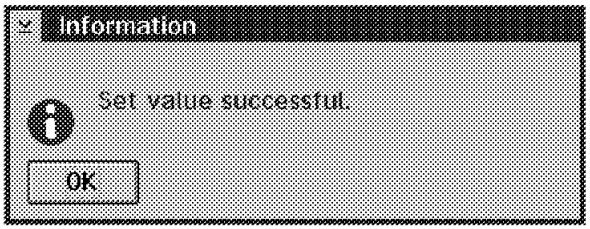


Figure 111. The Successful Completion Message

Note: Attempting to change the status of a communication object may result in an error, for example, if an attempt is made to vary off an active object. If the status of a communication object doesn't change following a succesful set operation, the QSYSOPR message queue and/or QHST log should be checked for errors. Figure 119 on page 177 is an example of such a message.

In Figure 112 on page 170 we can see that the APPC controller is now active - that the set operation was successful.

```
                        Work with Configuration Status                RALYAS4C
                                                              08/08/96  13:41:41
Position to  . . . . .               Starting characters

Type options, press Enter.
  1=Vary on   2=Vary off   5=Work with job   8=Work with description
  9=Display mode status ...

Opt  Description       Status               -------------Job--------------
     TRN2619C         ACTIVE
       RALYAS4B       ACTIVE
         RALYAS4B     ACTIVE









                                                                       Bottom
Parameters or command
===>
F3=Exit   F4=Prompt   F12=Cancel   F23=More options   F24=More keys
```

Figure 112. The Status after Setting the ibmappnNodeLsState Variable

9.4.2 Client Management MIB

The Client Management MIB subtree has four children which form the following groups:

clntSystem The Client System Group provides global information about the client, connectivity, and capabilities.

clntHardware The Client Hardware Group provides information that is related to storage, devices, file systems, and printers.

clntSoftware The Client Software Group provides information that is related to installed software and fixes.

The Client Management MIB is covered in detail in 8.1.2, “The Client Management MIB” on page 137.

RFC: None

RFC Noted Exception: None

MIB Subtree Description: Client Management

MIB Subtree Object Identifier: clientMgmtSubAgent ::= { internet(1) private(4) enterprise(1) ibm(2) ibmprod(6) 50 }

MIB Module Name: IBMCLTM

Prerequisite MIB Modules: RFC1155, RFC1212, IBMMIB ibm.mib

9.4.2.1 Browsing the Client Management MIB

For examples of browsing the Client Management MIB refer to 8.1.2, “The Client Management MIB” on page 137.

9.4.3 NetView for AIX Subagent MIB

One MIB object in this MIB is supported. The object provides the average percentage of load (processor utilization) during the elapsed time. Each retrieval of this value is calculated in the same manner as the value that is displayed by the DSPSYSSTS command when the *restart* function key is used.

RFC: None

RFC Noted Exception: None

MIB Subtree Description: NetView for AIX subagent ComputerSystem group

MIB Subtree Object Identifier: nv6saComputerSystem ::= { internet(1) private(4) enterprises(1) ibm(2) prod(6) netView6000SubAgent(4) 5 }

MIB Module Name: IBMNV6SA

The ASN.1 for this MIB is available in member IBMNV6SA in file QSYS/QANMMIB.

Prerequisite MIB Modules: RFC1155, RFC1212, IBMAS400

Note: A NetView for AIX application provides support for this data.

NetView for AIX Subagent MIB Object Supported

```
nv6saComputerSystemLoad OBJECT-TYPE

    SYNTAX  Gauge
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
    The CPU load as a percentage.  For example, 25% is 2500.
     ::= { nv6saComputerSystem 1 }
```

9.4.3.1 Browsing the NetView for AIX Subagent MIB

For examples of browsing the NetView for AIX Subagent MIB refer to 5.1.5, “The NetView for OS/2 Grapher Function” on page 57 where we refer to it as netView6000SubAgent MIB.

9.4.4 AS/400 Remote Workstation MIB

The OS/400 V3R6 SNMP enhancements include support for the AS/400 Remote Workstation (RWS) MIB. This MIB allows information about IBM 5494 remote workstation controllers attached to an AS/400 to be queried via SNMP. The MIB is located on the AS/400, not on the 5494. 5494 VPD (vital product data) information is retrieved from the 5494 and displayed via the MIB.

The IBM 5494 remote workstation controller is a hardware device that allows access from remote workstations (terminals, personal computers and printers) to an AS/400 system over either a LAN or WAN connection.

MIB Module Name: IBMRWS

MIB Subtree Object Identifier: ibmAs400Rws ::= { enterprises(2) ibm(6) ibmProd(11) ibmAs400Agent(1) }

Prerequisite MIB Modules: RFC1212, RFC1213, RFC1155

The AS/400 remote workstation MIB subtree has three children. These are:

- rwsTable
- rwsDevTable
- rwsGeneralInfo

rwsTable: This table provides information about 5494 remote workstation controllers attached to the managed AS/400. Browsing this subtree, you can see information such as the name of remote controller, its operational state, the name of the associated APPC controller and device, and information about remote controller itself (model, type, serial number and the microcode version). There are two read-write variables in this MIB group. These are rwsOperState, which relates to the operational state of the RWS controller in the managed AS/400, and rwsAppcDevState, which relates to the operational state of the APPC device associated with the remote workstation controller on the managed AS/400. A set operation performed against rwsOperState changes the operational status of the 5494 RWS controller on the managed AS/400. A set operation performed against rwsAppcDevState changes the operation status of the APPC device associated with the remote workstation controller on the managed AS/400. Note that the operational status of the APPC controller associated with the remote workstation controller can be queried and changed via the APPN MIB (the remote workstation controller is an APPN LEN node). See 9.4.1.2, “Using the APPN MIB set function to Manage AS/400 Lines and Controllers” on page 167.

rwsDevTable: This table provides information about devices (workstations) associated with 5494 remote workstation controllers attached to the managed AS/400. Browsing this subtree, you can see information such as the name of the devices, their operational status, their serial numbers and the name of the remote workstation controller the devices are attached to. There is one read-write capable variable in this MIB group. This is rwsDevOperState and it relates to the operational status of the device. A set operation performed against this variable changes the operation status of the 5494 RWS device on the managed AS/400.

rwsGeneralInfo: This MIB subtree has the only child, it represents the number of remote workstation controllers attached to the managed AS/400.

9.4.4.1 Browsing the AS/400 Remote Workstation MIB

Using a MIB browser on this group, we are able to see three kinds of information. These are:

- Information about remote workstation controllers attached to the managed AS/400.
- Information about devices attached to these remote workstation controllers.
- Information on the number of remote workstation controllers attached to the managed AS/400.

Before we can query the RWS MIB objects, we must load this enterprise MIB into NetView for OS/2. To do this we follow the steps shown in Chapter 13,

"Loading an Enterprise-Specific MIB" on page 219. The MIB to be loaded is called IBMRWS and is located in QSYS/QANMMIB.

rwsTable: First let's look at information related to the 5494 remote workstation controller itself. To do this follow this path:

enterprises→
ibm→
ibmprod→
ibmAs400Agent→
ibmAs400Rws→
rwsTable

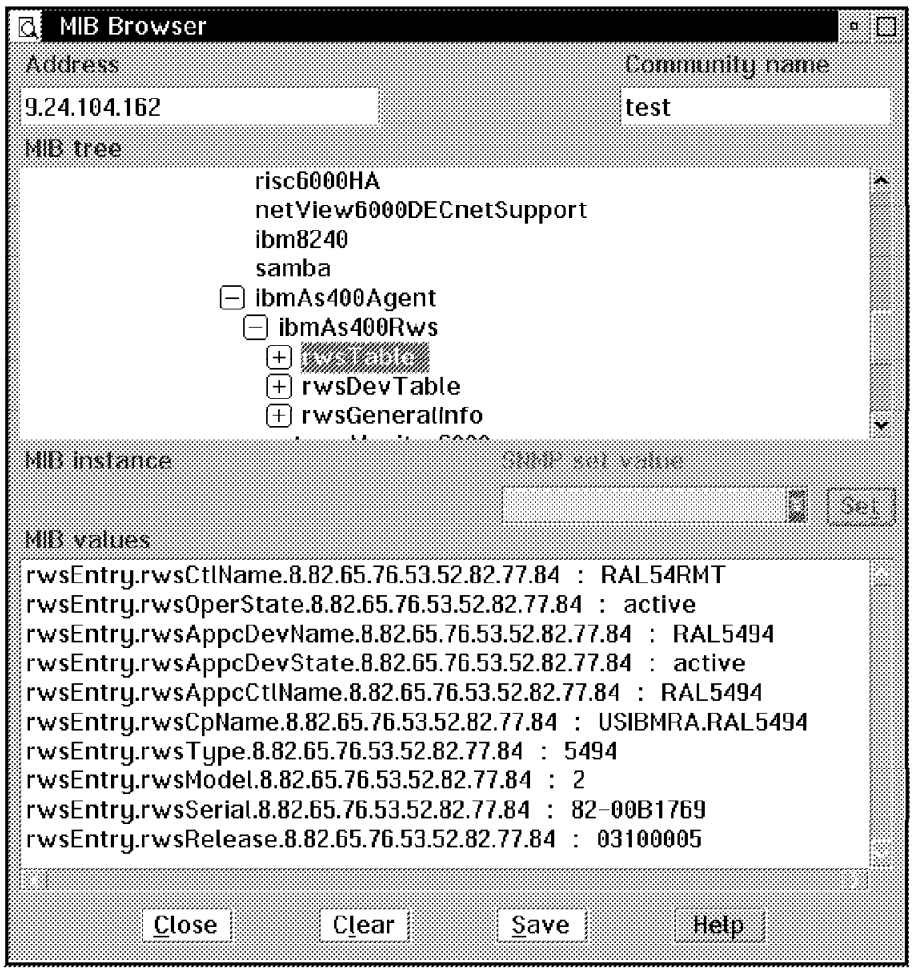


Figure 113. Results of a Query on the rwsTable MIB Group

In Figure 113 we can see the result of a query performed on the rwsTable MIB group. In this example only one 5494 was attached to the AS/400. Had multiple 5494s been attached, then each would be identified by a unique instance identifier. The query shows:

- That the name of the remote workstation controller (rwsCtlName) is RAL5494RMT.
- That its state (rwsOperState) is active.
- That the associated APPC device (rwsAppcDevName) is RAL5494 which is also active (rwsAppcDevState).

- That the associated APPC controller (rwsAppcCtlName) is also called RAL5494.
- That the 5494 fully qualified APPN name (rwsCpName) is USIBMRA.RAL5494.
- That the remote workstation controller type (rwsType) is 5494 and the model (rwsModel) is 2.
- That the 5494 serial number (rwsSerial) is 82-00B1769.
- That the microcode release (rwsRelease) is 03100005 (release 0310, modification 00, patch level 05).

rwsDevTable: Now let's look at information related to devices attached to the 5494 remote workstation controller. To do this follow this path:

enterprises→
ibm→
ibmprod→
ibmAs400Agent→
ibmAs400Rws→
rwsDevTable

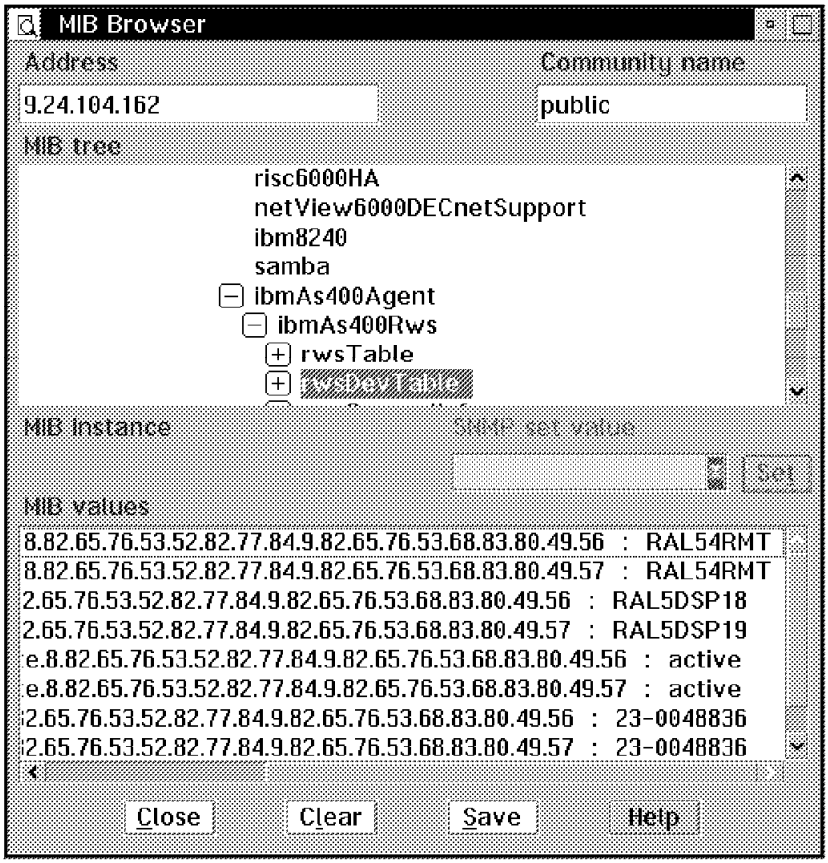


Figure 114. Results of Query on the rwsDevTable MIB Group

In Figure 114 we can see the result of a query performed on rwsDevTable MIB group. In this example two devices are attached to the 5494 each with a unique instance identifier. The query shows:

- That the name of the remote workstation controller (rwsDevCtlName) for both devices is RAL5494RMT.

- That the device names (rwsDevName) are RAL5DSP18 and RAL4DSP19 and that both are active (rwsDevOperState).
- That the device serial number (rwsDevSerial) in each case is 23-0048836 (this was a dual address display).

9.4.4.2 Using SNMP to Manage the Remote Workstation Controller

As we discussed in 9.4.4, “AS/400 Remote Workstation MIB” on page 171, there are three read-write MIB capable variables for the remote workstation MIB. These are:

- ibmAs400Rws.rwsTable.rwsEntry.rwsAppcDevState
- ibmAs400Rws.rwsTable.rwsEntry.rwsOperState
- ibmAs400Rws.rwsDevTable.rwsDevEntry.rwsDevOperState

In this example we will use the rwsAppcDevState MIB variable to activate (vary on) an inactive (varied off) 5494 APPC device description. In Figure 115 we can see that the 5494 APPC device state is currently varied off.

```
                        Work with Configuration Status                RALYAS4C
                                                              08/09/96  15:20:58
Position to  . . . . .               Starting characters

Type options, press Enter.
  1=Vary on   2=Vary off   5=Work with job   8=Work with description
  9=Display mode status ...

Opt  Description       Status                -------------Job--------------
     RAL5494          VARIED ON
       RAL5494        VARIED OFF









                                                                        Bottom
Parameters or command
===>
F3=Exit   F4=Prompt   F12=Cancel   F23=More options   F24=More keys
```

Figure 115. The Status before Setting the rwsAppcDevState Variable

The rwsAppcDevState MIB variable is located within the rwsTable. To locate the rwsTable, follow this path:

enterprises→
ibm→
ibmprod→
ibmAs400Agent→
ibmAs400Rws→
rwsTable

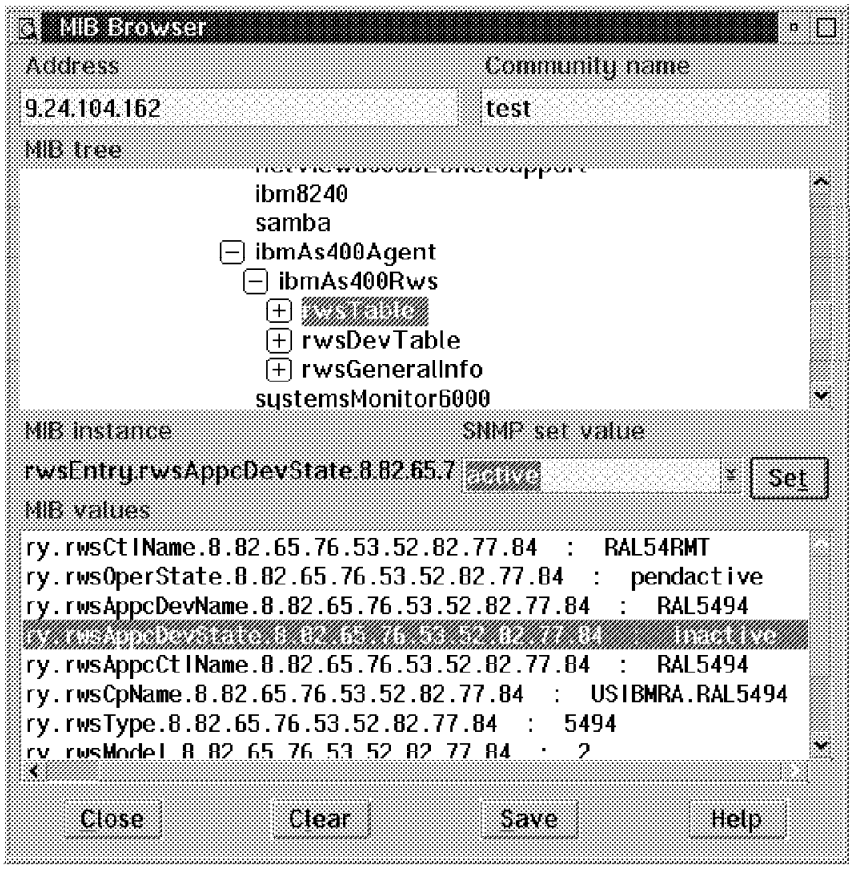


Figure 116. Setting the rwsAppcDevState MIB Variable

Having selected the rwsAppcDevState MIB variable for the appropriate remote workstation controller (via rwsAppcDevName), we select a set value of active from the SNMP set value window then press the **Set** button. Note that SNMP community *test* allows write access. If the set operation is successful you will see the completion message, as shown in Figure 117.

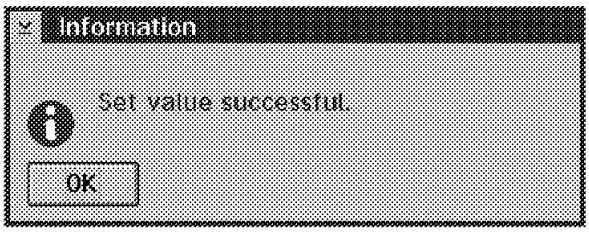


Figure 117. Completion Message

In Figure 118 on page 177 we can see that the 5494 APPC device is now active - that the set operation was successful.

```
                        Work with Configuration Status                RALYAS4C
                                                              08/09/96  15:23:47
Position to  . . . . .               Starting characters

Type options, press Enter.
  1=Vary on   2=Vary off   5=Work with job   8=Work with description
  9=Display mode status ...

Opt  Description       Status                -------------Job--------------
     TRN2619C         ACTIVE
       RAL5494        ACTIVE
         RAL5494      ACTIVE
           QRMTWSC    ACTIVE/TARGET         RAL5494     QUSER       000578
           QRMTWSC    ACTIVE/SOURCE         RAL5494     QUSER       000578




                                                                        Bottom
Parameters or command
===>
F3=Exit   F4=Prompt   F12=Cancel   F23=More options   F24=More keys
```

Figure 118. The Status after Setting the rwsAppcDevState Variable

Note: Attempting to change the status of a communication object may result in an error, for example, if an attempt is made to vary off an active object. If the status of a communication object doesn't change following a successful set operation, the QSYSOPR message queue and/or QHST log should be checked for errors. Figure 119 is an example of such a message.

```
                        Additional Message Information

Message ID . . . . . :   CPD2643
Date sent  . . . . . :   08/09/96      Time sent  . . . . . :   12:20:33

Message . . . . :   Device RAL5DSP18 not varied off.  Error occurred.

Cause . . . . . :   Device RAL5DSP18 was not varied off for one of the
  following reasons:
    --The device is active.
    --The job default wait time was not long enough to allow the vary off
  function to complete.
Recovery  . . . :   Do one of the following:
    --Try the request again when the device is not active.
    --Use the default maximum wait time (DFTWAIT parameter) in the Change Job
  (CHGJOB) command to increase the job default wait time.  Then try the Vary
  Configuration (VRYCFG) command again.

                                                                        Bottom
Press Enter to continue.

F1=Help   F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 119. Message CPD2643

9.5 Novell Enterprise MIBs

OS/400 V3R2 and V3R7 provided the AS/400 with IPX router support. Included in this support are MIBs that allow IPX, RIP, SAP and NLSP information on a managed AS/400 to be quered via SNMP. As an introduction to IPX, below are short descriptions of IPX, RIP, SAP and NLSP.

IPX IPX is the protocol that flows between systems in a Novell network or internetwork. RIP, SAP and NLSP are examples of upper layer protocols that use IPX as their underlying protocol.

RIP RIP is a distance-vector routing protocol used by IPX routers on the network. RIP allows a router to exchange routing information with a neighbor router. RIP routers periodically broadcast all known routing information, even if nothing has changed. This performance weakness is avoided by the NLSP protocol.

SAP SAP allows service-providing nodes, such as file servers and print servers, to advertise their services and addresses in order for clients to access these services. The SAP protocol is also used by clients to query available services in the network, the client can then directly request a service from a particular server. Like RIP, SAP is periodically broadcast even if nothing has changed. This performance weakness is avoided by the NLSP protocol.

NLSP NLSP is used by IPX routers to share their routing and service information with other devices on the network. NLSP provides better performance, scalability, reliability, and management of network traffic than the RIP and SAP protocols. RIP and SAP are noisy protocols in that they periodically broadcast the contents of their tables and hence need more bandwidth on a network for network protocol processing than the NLSP protocol.

9.5.1 Internetwork Packet Exchange (IPX) MIB

The IPX MIB defines the management information for a system using the IPX protocol. The MIB consist of four groups:

ipxSystem This group contains general information about all instances of IPX of the system.

ipxCircuit This group contains information about all circuits used by IPX on the system.

ipxForwarding This group contains generic routing information that must be provided by any system using the IPX routing protocol.

ipxServices This group contains information about all known services.

MIB Module Name: IPX.MIB

MIB Subtree Object Identifier: ipx ::= { enterprises(23) novell(2) mibdoc(5) }

Prerequisite MIB Modules: RFC1155, RFC1212, RFC1213, RFC1215

9.5.1.1 Browsing Internetwork Packet Exchange (IPX) MIB

Before we can query the IPX MIB objects, we must load this enterprise MIB into NetView for OS/2. To do this we follow the steps shown Chapter 13, “Loading an Enterprise-Specific MIB” on page 219. The MIB module can be obtained from the following sources:

- The FTP server at nisc.sri.com
- A Novell representative
- From the IANA FTP server at venera.isi.edu

See also the information on 9.4, “IBM Enterprise MIBs” on page 159.

Although Novell allows set functionality against MIB objects, the OS/400 implementation does not support the use of set against any IPX MIB object.

As an example of the information available via the IPX MIB and how this compares with information available via AS/400 system commands, let’s look at the AS/400 RIP and SAP tables via both methods. Looking first at the RIP table, this can be displayed at the AS/400 by entering the command WRKIPXSTS (*RTE).

```
                        Display IPX Route Information
                                                              System:   RALYAS4A
Type options, press Enter.
  5=Display details

         Remote        Number      Number      Next Hop         Route
Opt      Network      of Hops     of Ticks     Node Address     Source
         00000009         0            1      *NONE            *CCT
         30AA0F92         1            2      400052005240     *RIP
         31ECF1DD         1            2      08005A0D2860     *RIP
         53914BBA         1            2      08005A0D294A     *RIP
         5F7289F6         0            1      000000000001     *LOCAL
         760D0117         1            3      400052005010     *RIP

                                                                        Bottom
F3=Exit    F5=Refresh    F6=Print list    F12=Cancel    F17=Top    F18=Bottom
```

Figure 120. The Display IPX Route Information Screen

From Figure 120 we can see that there are six IPX networks known to the AS/400. The networks with a route source of *RIP were discovered via RIP. (The network with a route source of *CCT represents a network defined on the AS/400. The network with a route source of *LOCAL represents the AS/400’s internal network.)

Now let’s look at this same information via the IPX MIB. The following queries were performed on MIB variables in ipxDestEntry part of MIB subtree:

private→
enterprises→
novell→
mibdoc→

ipx→
ipxForwarding→
ipxDestTable→
ipxDestEntry

Figure 121 shows the results of a query on the ipxDestNetNum MIB variable.

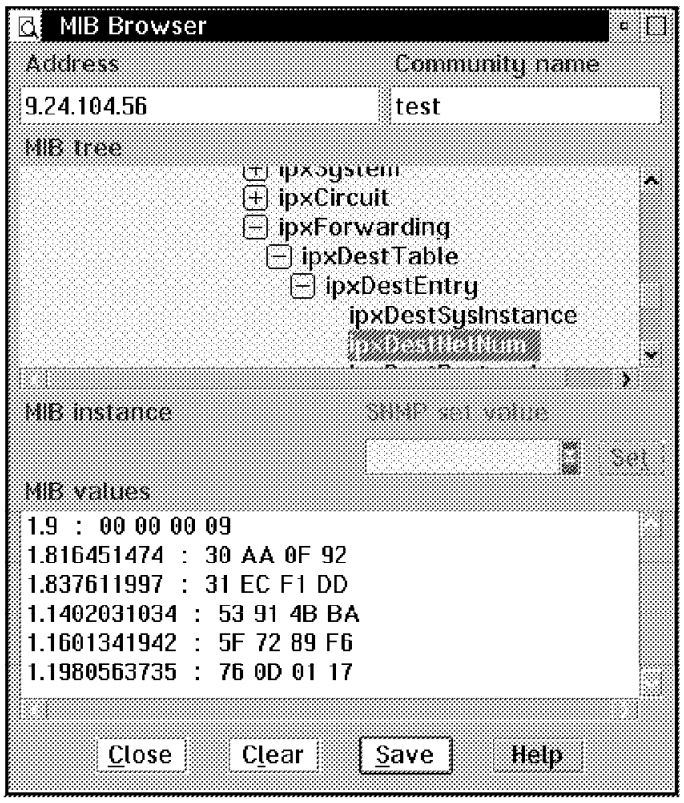


Figure 121. Results of Query on the ipxDestNetNum MIB Variable

As can be seen, the right column of variables represents the IPX network numbers in the same order as they are presented on a native AS/400 screen (Figure 120 on page 179).

Figure 122 shows the results of a query on the ipxDestHopCount MIB variable.

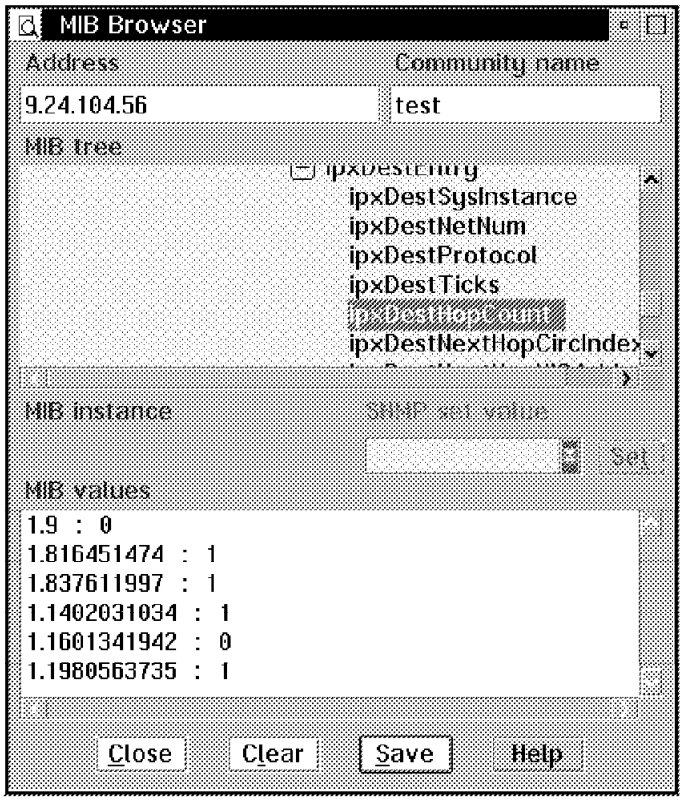


Figure 122. Results of Query on the ipxDestHopCount MIB Variable

As can be seen, the right column of variables represents the hop counts in the same order as they are presented on a native AS/400 screen (Figure 120 on page 179). The term hop count represents the number of intermediate routers between the local network and the destination network.

Figure 123 shows the results of a query on the ipxDestTicks MIB variable.

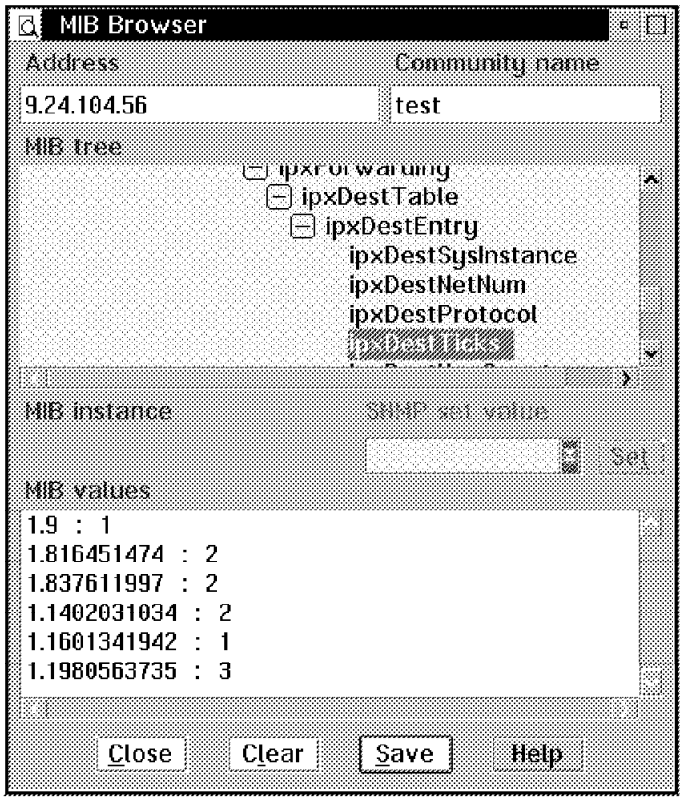


Figure 123. Results of Query on the ipxDestTick MIB Variable

As can be seen, the right column of variables represents the tick counts in the same order as they are presented on a native AS/400 screen (Figure 120 on page 179). A tick is equal to 1/18th of a second. It is used to compare route entries - if two routes entries have the same hop count, the route with the least tick count is used.

Figure 124 shows the results of a query on the ipxDestNextHopNetNum MIB variable.

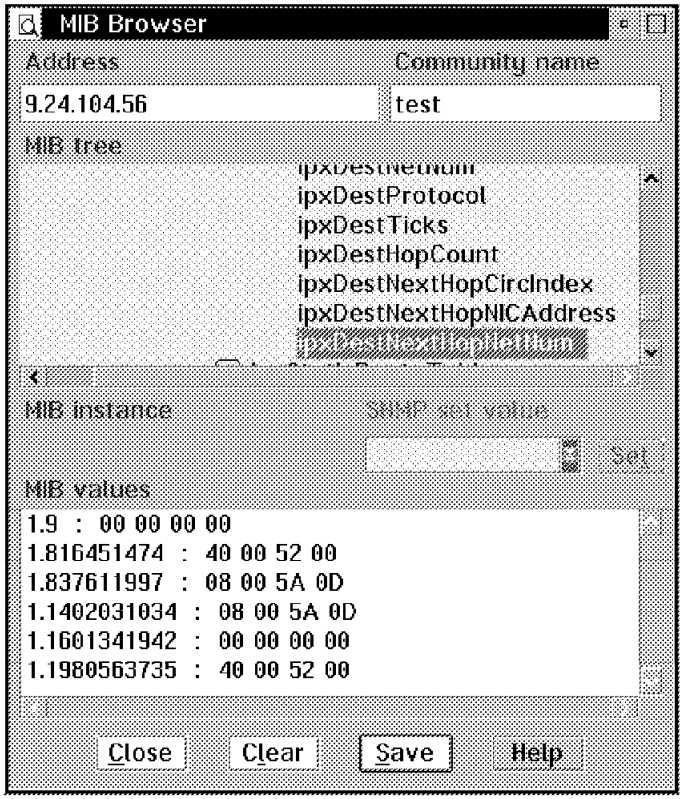


Figure 124. Results of Query on the ipxDestNextHopNetNum MIB Variable

As can be seen, the right column of variables represents the next hop node address in the same order as they are presented on a native AS/400 screen (Figure 120 on page 179).

Figure 125 shows the results of a query on the ipxDestProtocol MIB variable.

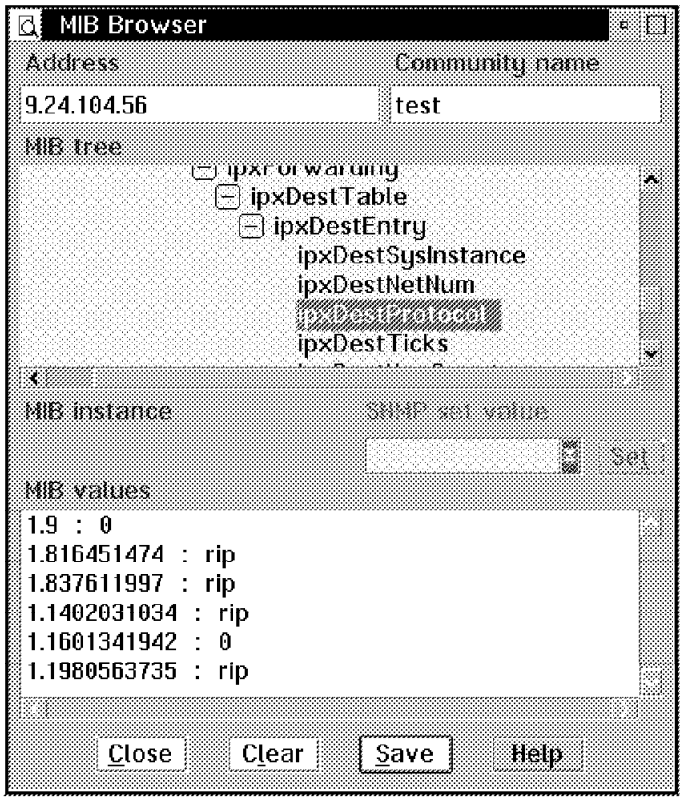


Figure 125. Results of Query on the ipxDestProtocol MIB Variable

As can be seen, the right column of variables represents the route source in the same order as they are presented on a native AS/400 screen (Figure 120 on page 179).

In the above examples we showed how information from the IPX MIB related to information from WRKIPXSTS (*RTE), IPX service information can also be obtained via the IPX MIB. Figure 126 shows how information from the IPX MIB relates to information from WRKIPXSTS (*SRV).

```
                        Display IPX Service Information
                                                              System:   RALYAS4A
Type options, press Enter.
  5=Display details

       Service                Service        Remote      Hops to    Service
Opt    name                   Type           Network     Service    Source
         1                      2              3           4          5
       FERGUS_NW41            *FILESVR       31ECF1DD       1       *SAP
       MANSERV1               *FILESVR       30AA0F92       1       *SAP
       NW41SERV               *FILESVR       53914BBA       1       *SAP
       BSER4.00-6.10_30A>     004B           30AA0F92       1       *SAP
       BSER4.00-6.10_539>     004B           53914BBA       1       *SAP
       ISMANSERV11841501>     0102           30AA0F92       1       *SAP
       ISNW41SERV0787289>     0102           53914BBA       1       *SAP
       IVMANSERV11841501>     0102           30AA0F92       1       *SAP
       IVNW41SERV0787289>     0102           53914BBA       1       *SAP
       LANPROTECT1841501>     0102           30AA0F92       1       *SAP
       LDPNVIRUS_PROTECT>     0102           53914BBA       1       *SAP
       FERGUS_NW41            0107           31ECF1DD       1       *SAP
       MANSERV1               0107           30AA0F92       1       *SAP
       NW41SERV               0107           53914BBA       1       *SAP
       FERGUS_NW41_____>      0115           31ECF1DD       1       *SAP
       FERGUS_NW41_____>      012B           31ECF1DD       1       *SAP
       FERGUS_NW41            0130           31ECF1DD       1       *SAP
       NW41SERV               0237           53914BBA       1       *SAP
       NW41SERV               0238           53914BBA       1       *SAP
       FERGUS_NW41_TREE       026B           31ECF1DD       1       *SAP
       NT4275R                026B           53914BBA       1       *SAP
       SYSMAN                 026B           30AA0F92       1       *SAP
       FERGUS_NW41_TREE       0278           31ECF1DD       1       *SAP
       NT4275R                0278           53914BBA       1       *SAP
       SYSMAN                 0278           30AA0F92       1       *SAP
       NW41SERV               027B           53914BBA       1       *SAP

                                                                     Bottom
F3=Exit   F5=Refresh   F6=Print list   F12=Cancel   F17=Top   F18=Bottom
```

Figure 126. The Display IPX Service Information Screen

Notes:

Figure 126 contains the IPX Service information. The same information can be displayed using SNMP by quering the following MIB objects:

1. ipxServName
2. ipxServType
3. ipxServNetNum
4. ipxServHopCount
5. ipxServProtocol

9.5.2 Routing Information and Service Advertising Protocols MIB

The RIP/SAP MIB defines the management information for the RIP and SAP protocols running in an IPX environment. It provides information in addition to the information provided by the IPX MIB.

MIB Module Name: RIPSAP.MIB

MIB Subtree Object Identifier: ripsap ::= { enterprises(23) novell(2) mibdoc(20) }

Prerequisite MIB Modules: RFC1155, RFC1212

The RIP/SAP MIB subtree contains the following MIB groups:

ripsapSystem This group contains global information about each instance of the RIP/SAP protocol running on the system.

ripsapCircuit This group contains information about all circuits used by IPX on the system.

ripCircTable This group contains a RIP information entry for each circuit known to the system.

All tables in this MIB are linked to an instance of IPX via the system instance identifier as defined in the IPX MIB.

9.5.2.1 Browsing the RIP/SAP MIB

As an example of the information available via the RIP/SAP MIB, let's look at the information made available by the ripsapSystem MIB object. The ripsapSystem MIB object can be located by following this path:

private→
enterprises→
novell→
mibdoc→
ripsap→
ripsapSystem

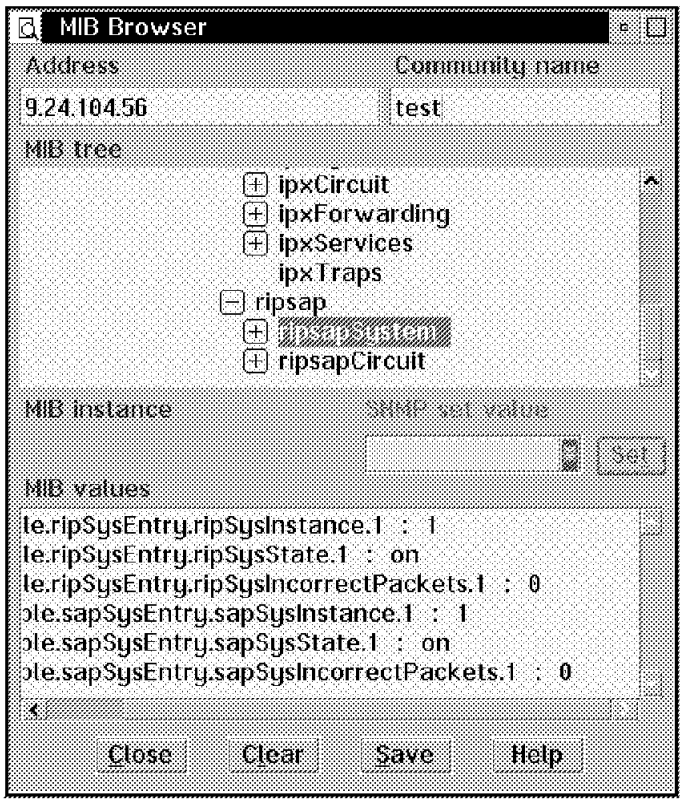


Figure 127. Results of Query on the ripsapSystem MIB Variable

The ripSysInstance and sapSysInstance entries provide index points to the table and correlate to ipxSysInstance within the IPX MIB information. In this example there is only one instance of each protocol running on the system. From ripSysState we can see that the RIP protocol is currently active and from ripSysIncorrectPackets we can see that there have been no incorrectly formatted RIP packets received. From sapSysState we can see that the SAP protocol is currently active and from sapSysIncorrectPackets we can see that there have been no incorrectly formatted SAP packets received.

9.5.3 NetWare Link Services Protocol (NLSP) MIB

The NLSP MIB defines the management information for the NLSP protocol running in an IPX environment. It provides information in addition to the information provided by the IPX MIB.

MIB Module Name: NLSP.MIB

MIB Subtree Object Identifier: nlsp ::= { enterprises(23) novell(2) mibdoc(19) }

Prerequisite MIB Modules: RFC1155, RFC1212, RFC1213, IPX.MIB

All tables in this MIB are linked to an instance of IPX via the system instance identifier as defined in the IPX MIB.

The NLSP MIB contains the following MIBs:

nlspSystem This group contains global information about each instance of NLSP running on one system.

nlspCircuit This group contains NLSP information for each IPX circuit known to this system.

nlspForwarding This group contains NLSP forwarding information in addition to that contained in the IPX forwarding group.

nlspNeighbors This group contains management information for each neighboring NLSP router known to the system.

nlspTranslation The translation group contains tables providing mappings between network numbers, NLSP system IDs, and router names.

nlspGraph The Graph group provides a representation of the network topology.

nlspLSP This table allows the path(s) that a packet may take to reach a destination to be reconstructed. The entries in this table represent those links that are one hop closer to the source and would be used for the minimum cost path(s) to reach the destination.

Work in Progress

At the time of writting this MIB was not fully implemented on the AS/400. We did not, therefore, browse the MIB contents.

Chapter 10. Host Resources MIB

The host resources MIB subtree has six children which form the following groups:

hrSystem This MIB group provides system information such as system up time, number of users and system date.

hrStorage This MIB group provides system storage information such as system memory size and DASD space available.

hrDevice This MIB group provides information on devices attached to the system such as whether a disk unit is removable and the status of a printer.

hrSWRun This MIB group provides information on the software running on the system such as the name and level of the software running.

hrRunPerf This MIB group provides information on the system's CPU performance. This MIB group is not supported by the AS/400.

hrSWInstalled This MIB group provides information on the software installed on the system such as a table of the software installed and when the software was last updated.

RFC: 1514

MIB Subtree Object Identifier: host ::= { mib-2 25 }

Prerequisite MIB Modules: RFC1212, RFC1213, RFC1155

The Host Resources MIB subtree: The host resources MIB is the mib-2 group MIB. Thus the subtree OBJECT IDENTIFIER for the host resources MIB is located under the mib-2 subtree. The OBJECT IDENTIFIER is
iso.org.dod.internet.mgmt.mib-2.host
or, in concise form: 1.3.6.1.2.1.25

In addition to being implemented by V3R6, the host resources MIB is implemented by Client Access Optimized for OS/2 and by the NetView for OS/2 agent. In the following pages we look at the AS/400 implementation and the Client Access Optimized for OS/2 implementation.

10.1 Host Resources MIB and AS/400

The OS/400 V3R6 SNMP enhancements include the addition of support for the host resources MIB. Support for the MIB is available via OS/400 V3R6 PTF SF28306. It is included in base V3R7.

Although the following six MIB objects are defined in RFC1514 as read-write capable, the current AS/400 implementation does not allow these variables to be set. The six objects are:

- hrSystemDate
- hrFSlastFullBackupDate
- hrFSlastPartialBackupDate
- hrSystemInitialLoadDevice
- hrSystemInitialLoadParametrs
- hrStorageSize

The following host resources MIB objects are not implemented by the AS/400:

- hrStorage.hrStorageTypes
- hStorageTable.hrStorageEntry.hrStorageAllocationFailures
- hrDevice.hrDeviceTypes
- hrDevice.hrDeviceTable.hrDeviceEntry.hrDeviceErrors
- hrDevice.hrNetworkTable
- hrDevice.hrPrinterTable
- hrDevice.hrFSTable.hrFSEntry.hrFSRemoteMountPoint
- hr.device.hrFSTypes
- hrSWRun.hrSWRunTable.hrSWRunEntry.hrSWRunPath
- hrSWRun.hrSWRunTable.hrSWRunEntry.hrSWRunParameters
- hrSWRunPerf

10.1.1 Browsing the Host Resources MIB on AS/400

Let's now run some queries on the objects within the host resources MIB group. Follow this path to locate the host MIB subtree:

mgmt→
mib-2→
host

hrSystem: In Figure 128 you can see the results of a query performed on the MIB group hrSystem.

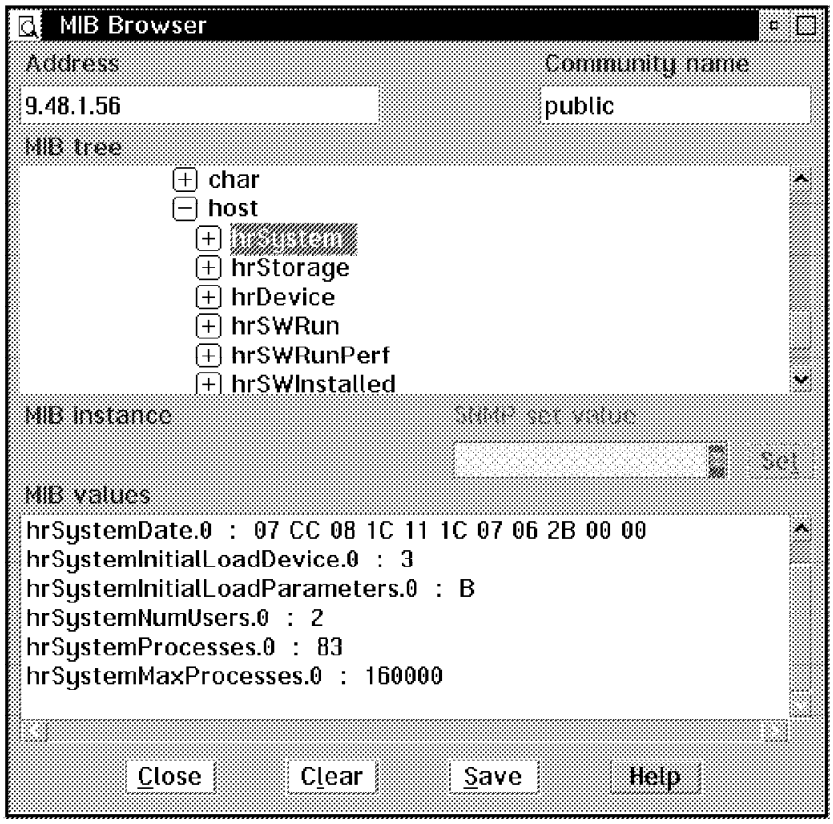


Figure 128. Results of Query on the hrSystem MIB Variable

From Figure 128 we can see: that the IPL was performed from device 3 (MIB group hrDevice could be queried for information on device 3), that the IPL was from load source area B (hrSystemInitialLoadParameters) and that the number of users at the time the picture was taken was 2.

hrStorage: In Figure 129 you can see the results of a query performed on the MIB group hrStorage. In this example we have used the MIB browser Save button save the results to a file.

```
MIB values :
hrMemorySize.0  :  131072
hrStorageIndex.1  :  1
hrStorageIndex.2  :  2
hrStorageIndex.3  :  3
hrStorageIndex.4  :  4
hrStorageIndex.5  :  5
hrStorageType.1  :  .iso.org.dod.internet.mgmt.mib-2.host.hrStorage.hrStorageTypes.hrStorageFixedDisk
hrStorageType.2  :  .iso.org.dod.internet.mgmt.mib-2.host.hrStorage.hrStorageTypes.hrStorageRam
hrStorageType.3  :  .iso.org.dod.internet.mgmt.mib-2.host.hrStorage.hrStorageTypes.hrStorageRam
hrStorageType.4  :  .iso.org.dod.internet.mgmt.mib-2.host.hrStorage.hrStorageTypes.hrStorageRam
hrStorageType.5  :  .iso.org.dod.internet.mgmt.mib-2.host.hrStorage.hrStorageTypes.hrStorageRam
hrStorageDescr.1  :  System ASP
hrStorageDescr.2  :  RAM
hrStorageDescr.3  :  RAM
hrStorageDescr.4  :  RAM
hrStorageDescr.5  :  RAM
hrStorageAllocationUnits.1  :  4096
hrStorageAllocationUnits.2  :  4096
hrStorageAllocationUnits.3  :  4096
hrStorageAllocationUnits.4  :  4096
hrStorageAllocationUnits.5  :  4096
hrStorageSize.1  :  2401280
hrStorageSize.2  :  12048
hrStorageSize.3  :  20583
hrStorageSize.4  :  73
hrStorageSize.5  :  64
hrStorageUsed.1  :  2048682
hrStorageUsed.2  :  7540
hrStorageUsed.3  :  7848
hrStorageUsed.4  :  64
hrStorageUsed.5  :  344
```

Figure 129. Results of Query on the hrStorage MIB Variable

In Figure 129 we can see information on the AS/400 storage units. The first row, hrMemorySize, provides information on the total random access memory on the system (in KB), that is in our case approximately 131 MB. The items with instance values 2-5 belong to random access memory while instance 1 is the system ASP, as can be seen from hrStorageDescr. Instances 2 to 5 represent memory pools on the AS/400. A memory pool is a dedicated part of random access memory concurrently used by a group of jobs. The hrStorageAllocationUnits section provides information on the storage allocation unit size, that is, 4,096KB for both memory and disk access. The hrStorageSize and hrStorageUsed values provide information similar to that available via the WRKSYSSTS command.

hrDevice: In Figure 130 and Figure 131 on page 194 you can see the results of queries performed on MIB group hrDevice. We have chosen to look at the hrDeviceDescr and hrDeviceStatus MIB objects. Both objects can be found in the host MIB subtree by selecting:

host→
hrDevice→
hrDeviceTable→
hrDeviceEntry

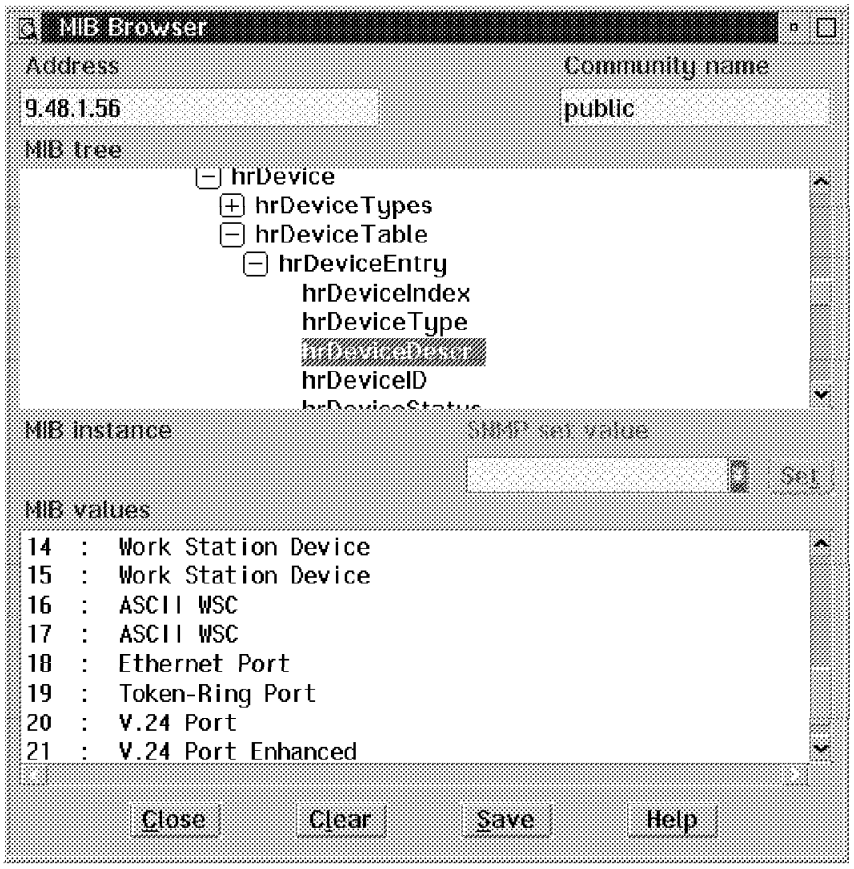


Figure 130. Results of Query on the hrDeviceDescr MIB Variable

In Figure 130 we can see a small part of the list of the hardware devices installed on the system located at 9.48.1.56. In Figure 131 on page 194 we can see the status of these same devices.

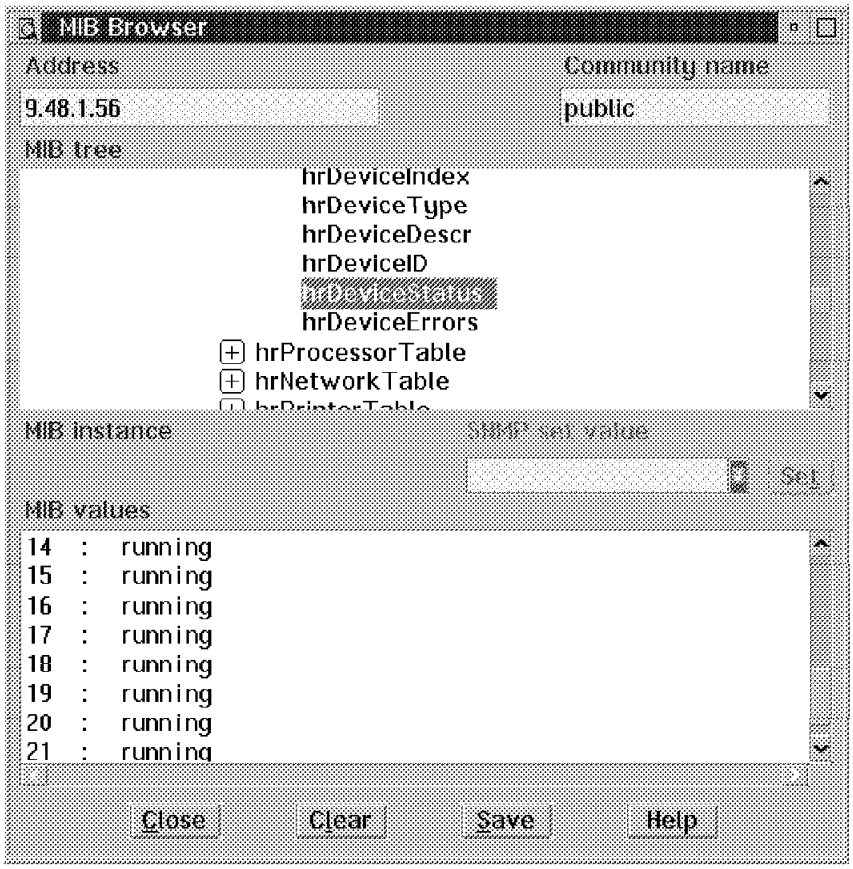


Figure 131. Results of Query on the hrDeviceStatus MIB Variable

The status can be unknown, running, warning, testing or down. A status of testing indicates that a diagnostic test is being performed on the device. A status of running means that the device is usable, while a status of down means that the device has a hardware problem. A status of unknown indicates that the device was presented previously on the system but now it is missing.

hrSWRun: In Figure 132 and Figure 133 on page 196 you can see the results of queries performed on MIB group hrSWRun. We have chosen to look at the hrSWRunName and hrSWRunStatus MIB variables. Both variables can be found in the host MIB subtree by selecting:

host→
hrSWRun→
hrSWRunTable→
hrSWRunEntry

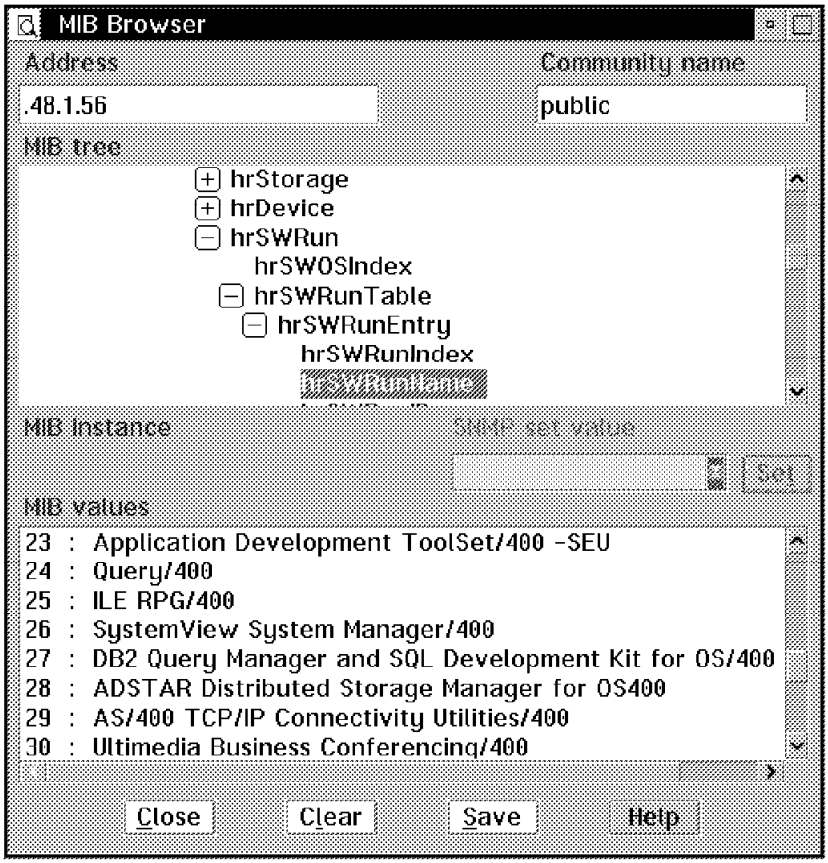


Figure 132. Results of Query on the hrSWRunName MIB Variable

In Figure 132 we can see a small part of the list of software running on the system located at 9.48.1.56. In Figure 133 on page 196 we can see the status of these same software programs.

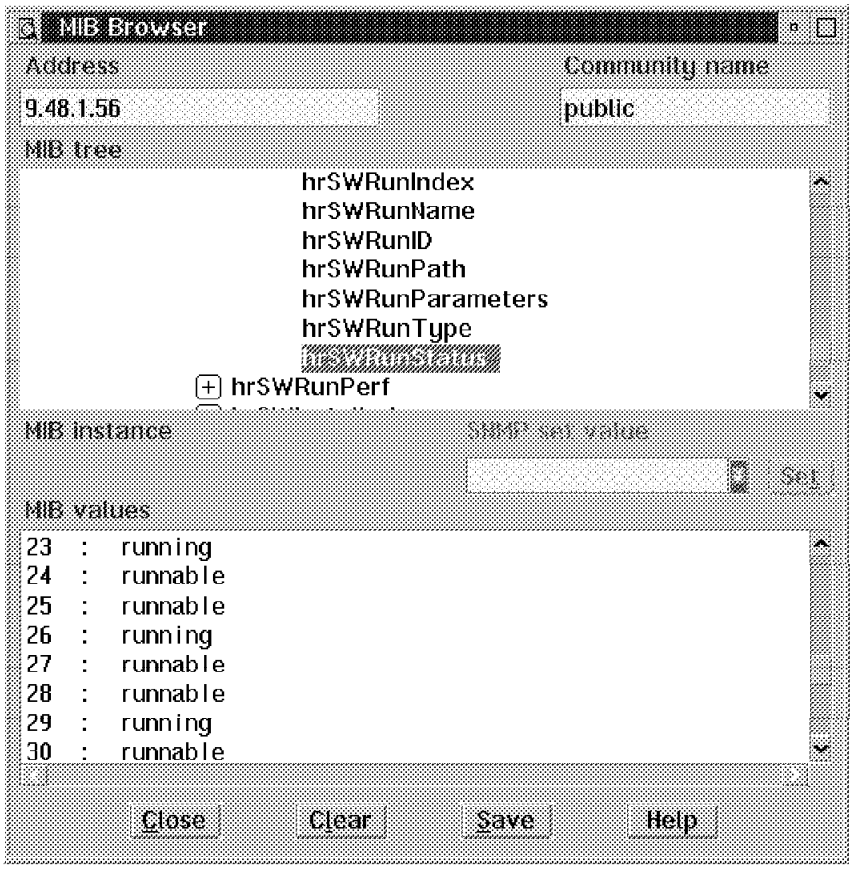


Figure 133. Results of Query on the hrSWRunStatus MIB Variable

From Figure 132 on page 195 and Figure 133 we are able to see, for example, that the system at address 9.48.1.56 has the Query/400 licensed program loaded and that is runnable; and that the Application Development Tool Set/400 is loaded and is currently in use (has a status of running).

hrRunPerf: At the time of writing hrRunPerf was not implemented by SNMP on the AS/400. System performance information can, however, be retrieved via the NetView for AIX subagent MIB. See 9.4.3, “NetView for AIX Subagent MIB” on page 171.

hrSWInstalled: In Figure 134 you can see the results of a query performed on the hrSWInstalledName. The hrSWInstalledName MIB can be reached by selecting:

host→
hrSWInstalled→
hrSWInstalledTable→
hrSWInstalledEntry→
hrSWinstalledName

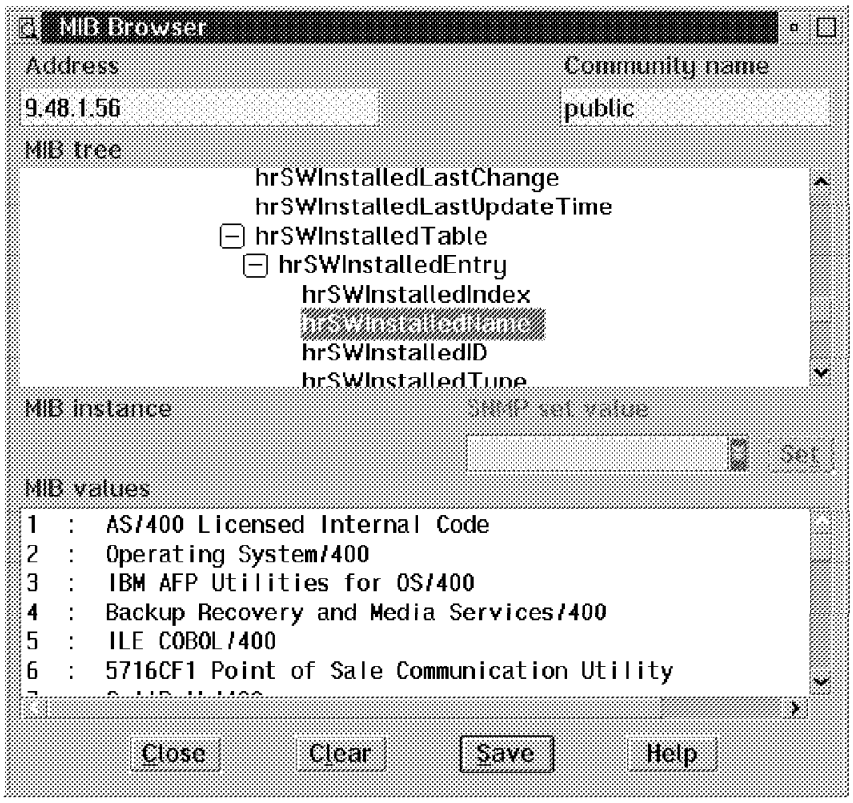


Figure 134. Results of Query on the hrSWinstalledName MIB Variable

In Figure 134 we can see a list of software installed on the system at 9.48.1.56. This list is as would be shown by option 10 of the command GO LICPGM. Other information is also available, such as the software type (hrSWInstalledType).

10.2 Host Resources MIB and Client Access Optimized for OS/2

Client Access Optimized for OS/2 also supports the host resources MIB. The implementation of the host resources MIB in Client Access Optimized for OS/2 is primarily intended to support the client inventory management functions of the AS/400 (see 8.1, “Client Inventory Management” on page 127) but can also be accessed directly from an SNMP manager. To activate the host resources MIB support on a personal computer running Client Access Optimized for OS/2 you are required to:

- Have configured Client Access using a protocol supported by SNMP on both managing and managed systems (native TCP/IP or AnyNet Sockets over SNA).

- Have enabled SNMP functions on the personal computer. To fulfill this task you will have to remove two REM commands in the CASERV.CMD file. The commands to be un-commented are: CALL CASNMP.CMD and DETACH SIASTART.CMD. See the *Inside Client Access/400 Optimized for OS/2*, SG24-2587 redbook for more information.

  Finally, you have to know what the community name to use. The AS/400 client management functions use a community name of IBMDMI. This can be seen if you perform a query of the client management MIB on the AS/400 that the Client Access system is connected as shown in Figure 135. The path to follow to locate this information is:

  private→
  enterprises→
  ibm→
  ibmprod→
  clientMgmtSubAgent→
  clntSystem

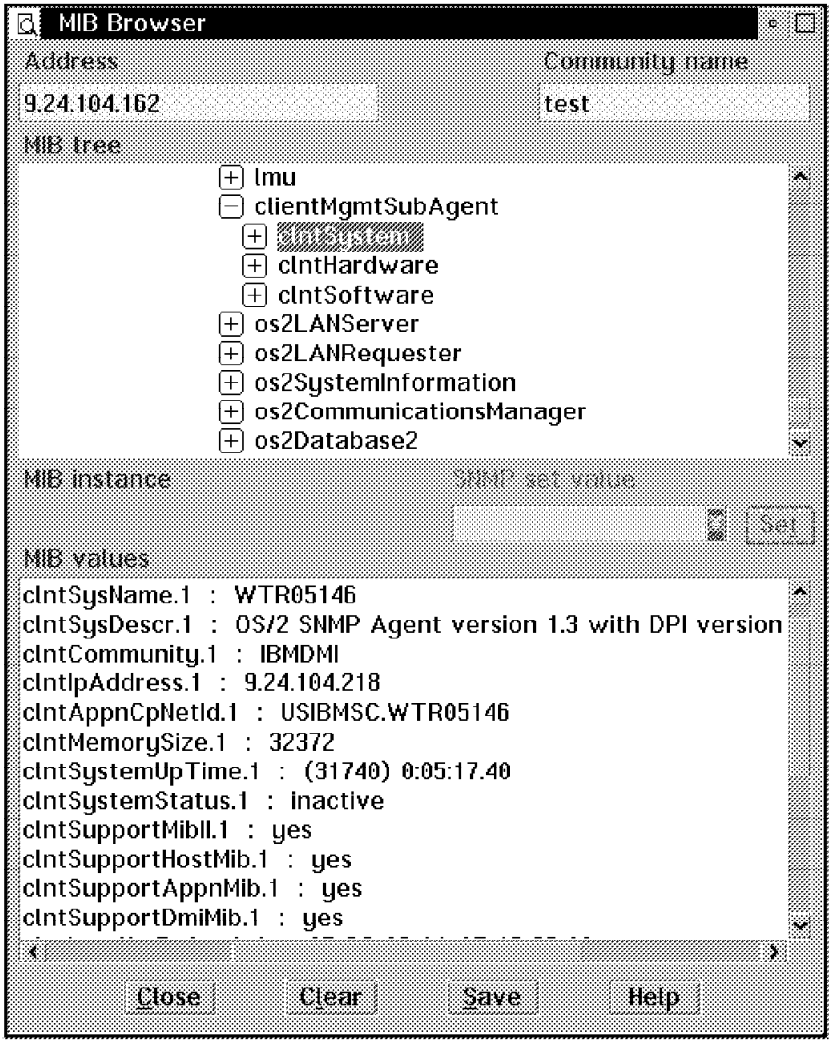


Figure 135. Results of Query on the clntSystem MIB Variable

In Figure 135 we can see the result of a query of the clntSystem MIB group. From this we can see that the client is using a community name (clntCommunity) of IBMDMI. This is the community name that we have to use when querying the

host resources MIB on the Client Access system. We can also see from Figure 135 that, in addition to supporting the host resources MIB (clntSupportHostMib), this client supports the APPN MIB (clntSupportAppnMib) and the DMI MIB (clntSupportDmiMib).

10.2.1 Browsing the Client Access Host Resources MIB

Let's now look at some examples of the information made available by the Client Access Optimized for OS/2 host resources MIB.

hrSystem: The hrSystem MIB group can be reached by selecting:

mib2→
host→
hrSystem

Figure 136 shows the results of a query performed on the hrSystem MIB object.

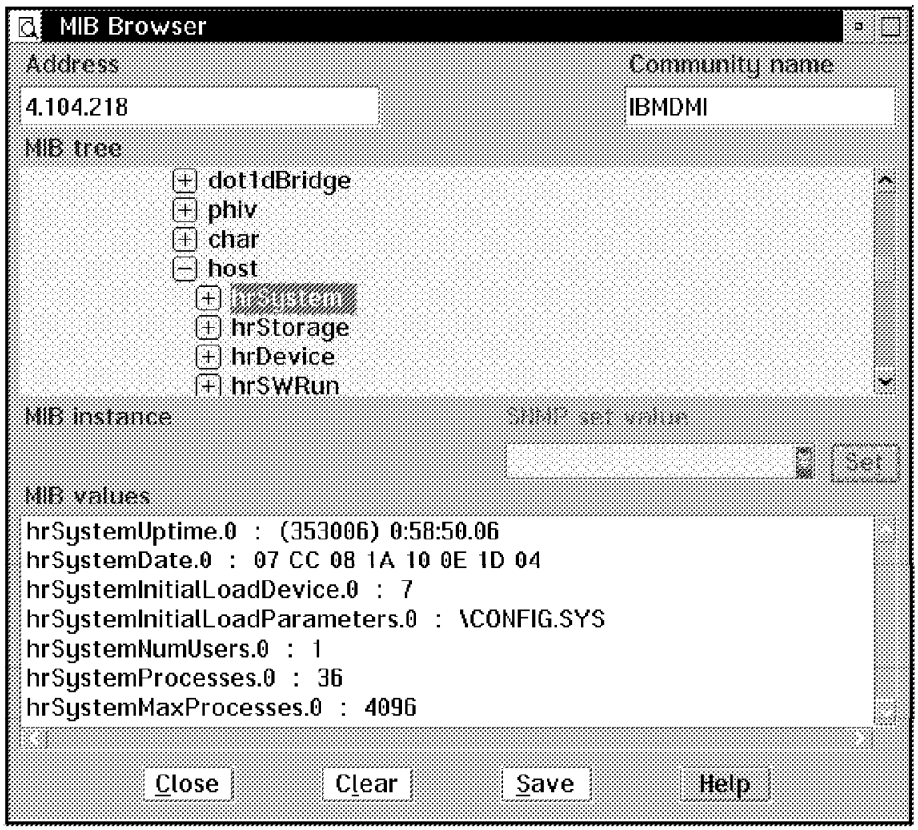


Figure 136. Results of Query on the hrSystem MIB Group

From Figure 136 we can see:

- That the personal computer has been running 58 minutes.
- That the PC boots from device 7 (MIB group hrDevice could be queried for information on device 7).
- That the IPL parameters are taken from CONFIG.SYS.
- That the system has just one user.
- That the current number of jobs running is 36 and that the maximum number of jobs that can be run is 4096.

hrDevice: In Figure 137, Figure 138 on page 201 and Figure 139 on page 202 you can see the results of queries performed on MIB group hrDevice. We have chosen to look at the hrDeviceDescr, hrDeviceStatus and hrPartitionTable MIB objects. These objects can be found in the host MIB subtree by selecting:

host→
hrDevice

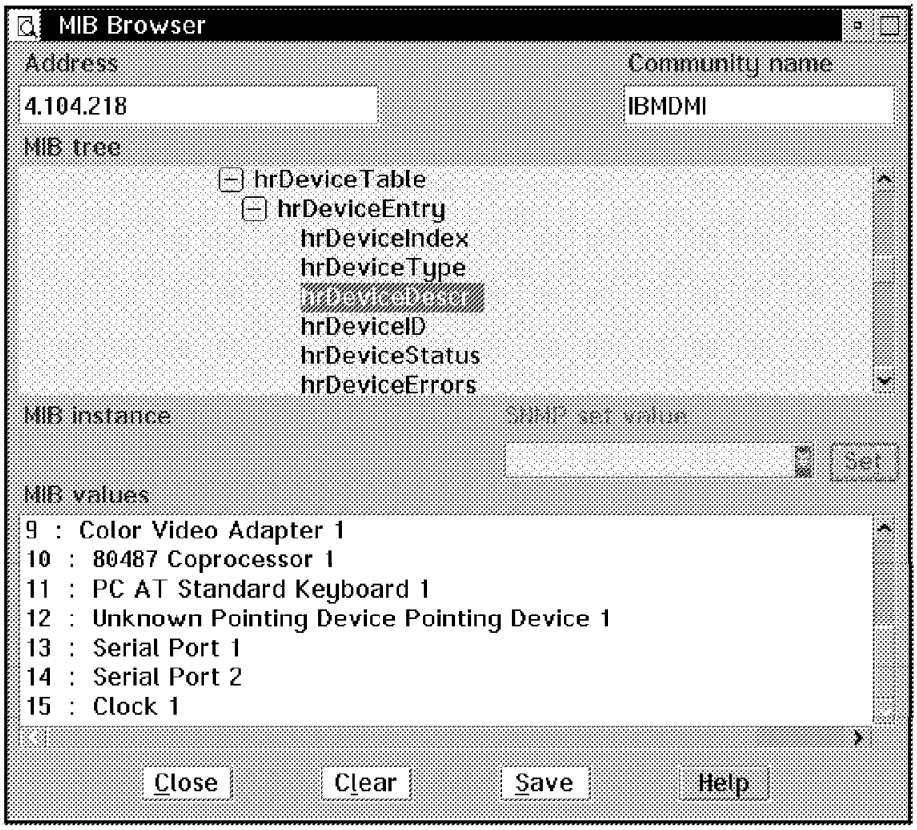


Figure 137. Results of Query on the hrDeviceDescr MIB Variable

In Figure 137 we can see a small part of the list of the hardware devices installed on the system located at 9.24.104.218. In Figure 138 on page 201 we can see the status of these same devices.

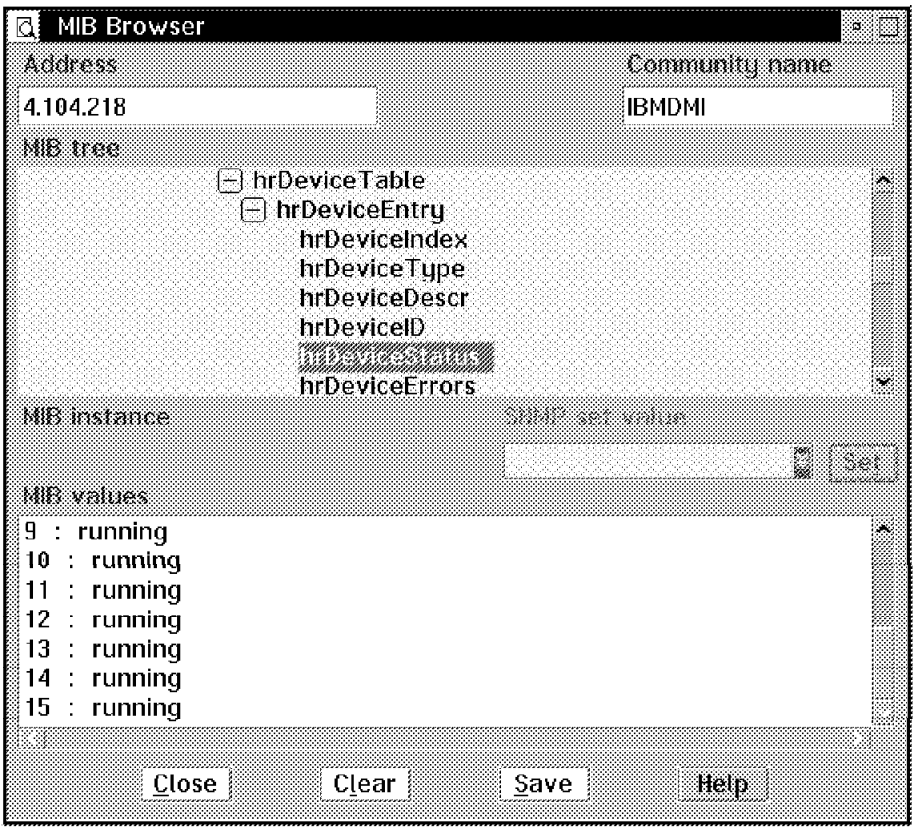


Figure 138. Results of Query on the hrDeviceStatus MIB Variable

In Figure 138 we can see that all listed devices are currently running (active).

In Figure 139 we can see the client system partition table.

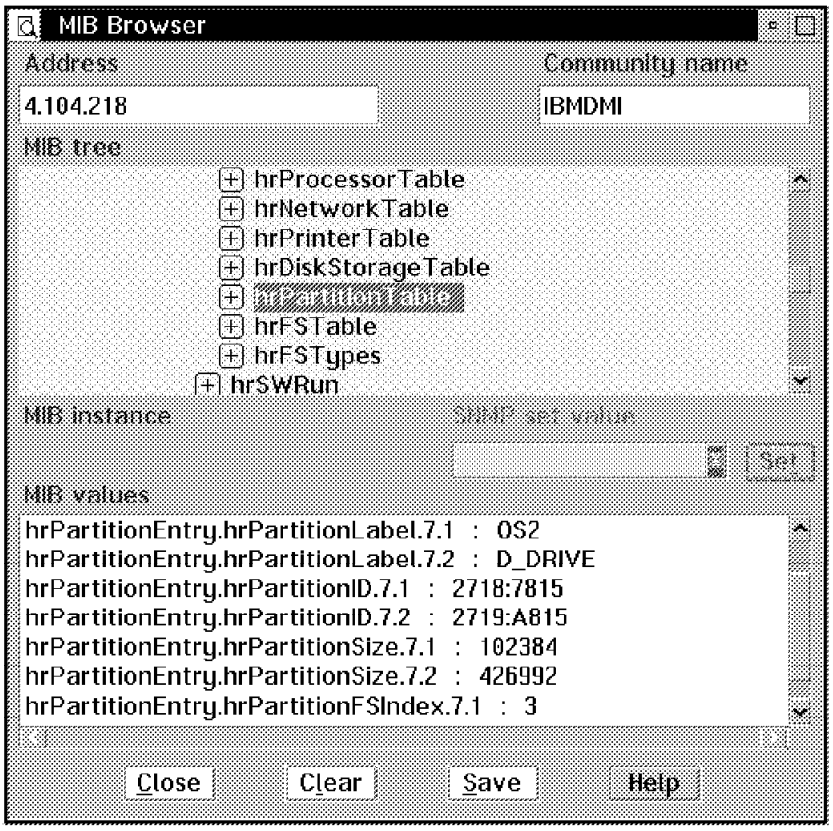


Figure 139. Results of Query on the hrPartitionTable MIB Group

From Figure 139 we can see, for example:

- That the PC disk has two partitions.
- That the volume labels associated with these partitions are OS2 and D_DRIVE.
- That the partition sizes are 102384 KB and 426992 KB.

This information is similar to that available via the FDISK and DIR commands.

hrSWRun: In Figure 140 you can see the result of a query performed on the MIB group hrSWRun. We have chosen to look at the hrSWRunName MIB variable. The variable can be found in the host MIB subtree by selecting:

host→
hrSWRun→
hrSWRunTable→
hrSWRunEntry→
hrSWRunName

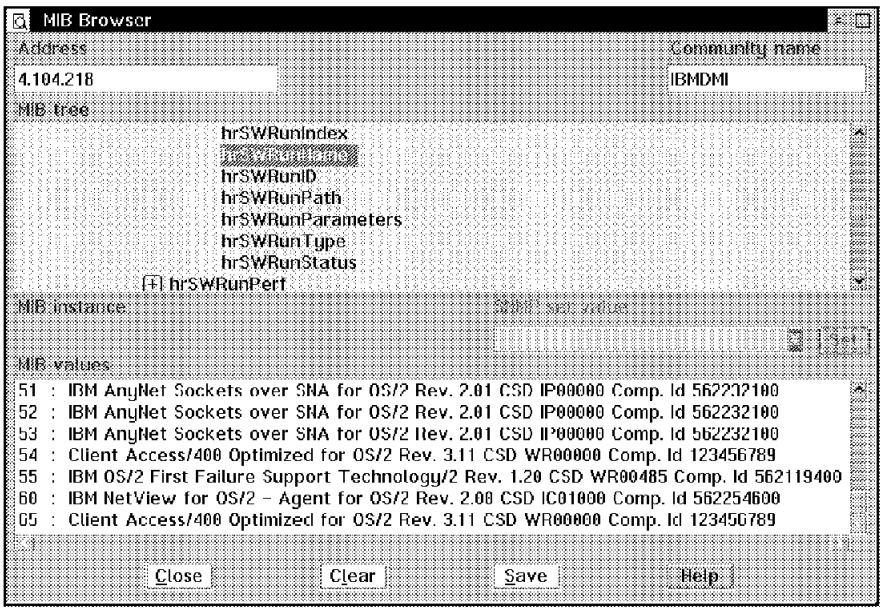


Figure 140. Results of Query on the hrSWRunName MIB Variable

In Figure 140 we can see a partial list of the software installed on the Client Access system. We can see, for example, that the IBM AnyNet Sockets over SNA for OS/2 product is installed, that its component ID 562232100, its revision is 2.01 and CSD is IP00000. This information would be useful when trying to resolve a software problem on the PC.

Chapter 11. AS/400 as a Trap Generator

As described in 3.1.6, “Trap Support” on page 27, the AS/400 can be a source of the following different types of traps:

- ColdStart
- WarmStart
- LinkDown
- LinkUp
- AuthenticationFailure
- EnterpriseSpecific

Traps are unsolicited data messages sent by an SNMP agent to its SNMP managing system. These messages are usually used to inform the managing station about a special condition that has occurred either in an agent system or in the network.

11.1 Configuration Requirements

For the AS/400 to generate a trap, the following configuration requirements must be in place:

- TCP/IP must be configured and running.
- The SNMP TCP/IP agent server job (STRTCPSVR SERVER(*SNMP)) must be running.
- SNMP must be configured to send traps.

To configure SNMP to send traps use the CHGSNMPA command.

```
                      Change SNMP Attributes (CHGSNMPA)

Type choices, press Enter.

Trap managers:
  Manager internet address . . .   '9.24.104.39'  1
  Community name . . . . . .       'PUBLIC'      2


  Translate community name . . .   *YES   3         *YES, *NO
              + for more values

                                                                      Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 141. The CHGSNMPA Command

Note:

1 Is the address of the SNMP manager to whom the traps should be sent.

2 Is the community name to be used for the traps.

3 Specifies the character set is to be used for the coding of community name on the managing system. In most cases this parameter should be set to *YES.

There are no special configuration requirements for NetView for OS/2 to accept traps other than that TCP/IP be supported as a management protocol. NetView for OS/2 does not care about the community name of incoming traps.

11.2 Examples of the AS/400 Generated Traps

In this section we will look at some examples of AS/400 generated traps using the NetView for OS/2 Event Displayer. The NetView for OS/2 Event Displayer is found in the NetView for OS/2 folder. Once started incoming traps can be displayed using the Display button.

11.2.1 The ColdStart Trap

The cold start trap in Figure 142 was generated when a TCP/IP SNMP agent was started on the AS/400 using the command STRTCPSVR SERVER(*SNMP). Message TCP4016 shown in Figure 143 on page 207 was also generated.

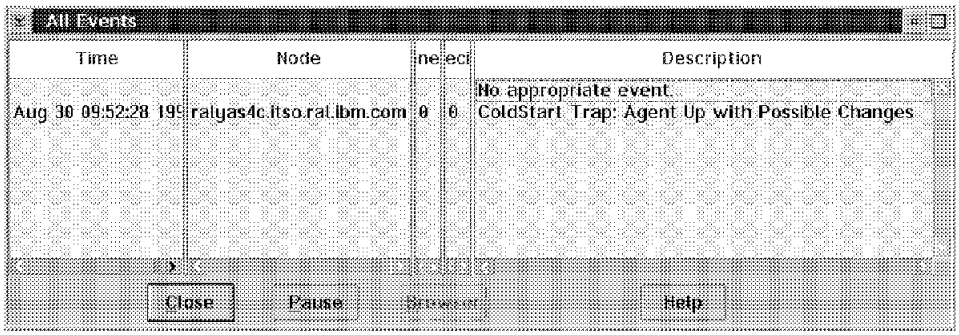


Figure 142. The ColdStart Trap

The NetView for OS/2 ColdStart trap in Figure 142 provides the following information to the administrator:

- The date and time the trap was logged in the trap log file.
- The node that generated the trap.
- The generic trap number 0, which indicates a ColdStart trap.
- The specific trap number 0.
- A text trap description.

```
                        Additional Message Information

Message ID . . . . . :   TCP4016
Date sent  . . . . . :   08/30/96      Time sent  . . . . . :   09:54:33

Message . . . . :   SNMP coldStart trap generated.

Cause . . . . . :   The conditions which cause this trap to be generated have
  occurred. An SNMP trap message has not necessarily been sent to an SNMP
  manager.

                                                                        Bottom
Press Enter to continue.

F1=Help   F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 143. The TCP4016 Message (ColdStart)

11.2.2 LinkDown and LinkUp Traps

The LinkDown and LinkUp traps in Figure 144 were generated when an AS/400 TCP/IP interface was stopped then restarted using the commands ENDTCPIFC ('9.24.102.198') and STRTCPIFC ('9.24.102.198'). Message TCP4016 shown in Figure 145 on page 208 was also generated. Note that this was not the TCP/IP interface via which the SNMP manager is reached.

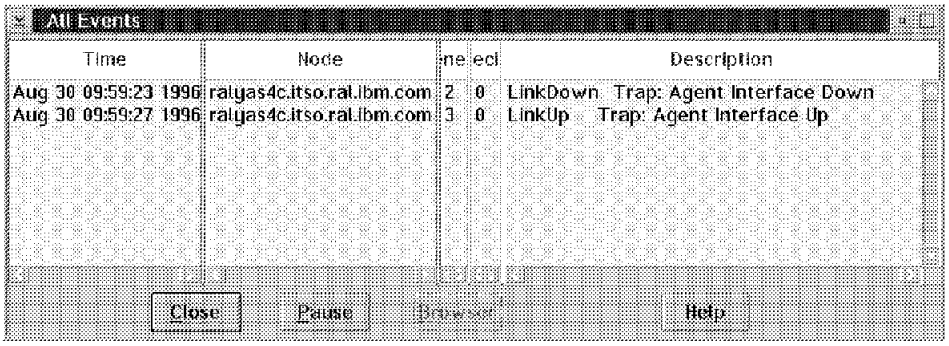


Figure 144. The LinkDown and LinkUp Traps

In Figure 144 we can see the generic trap number 2 indicating a LinkDown trap and the generic trap number 3 indicating a LinkUp trap.

```
                        Additional Message Information

Message ID . . . . . :   TCP4016
Date sent  . . . . . :   08/30/96      Time sent  . . . . . :   09:58:30

Message . . . . :   SNMP linkDown trap generated.

Cause . . . . . :   The conditions which cause this trap to be generated have
  occurred. An SNMP trap message has not necessarily been sent to an SNMP
  manager.









                                                                        Bottom
Press Enter to continue.

F1=Help   F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 145. The TCP4016 Message (LinkDown)

11.2.3 The AuthenticationFailure Trap

The AuthenticationFailure trap in Figure 147 on page 209 was generated by using a non-existing community name (BADCOMMUNITY) from a Web browser as shown in Figure 146 on page 209. The AS/400 must be enabled to send authentication traps using the command CHGSNMPA SNDAUTTRP(*YES) before authentication traps will be generated.

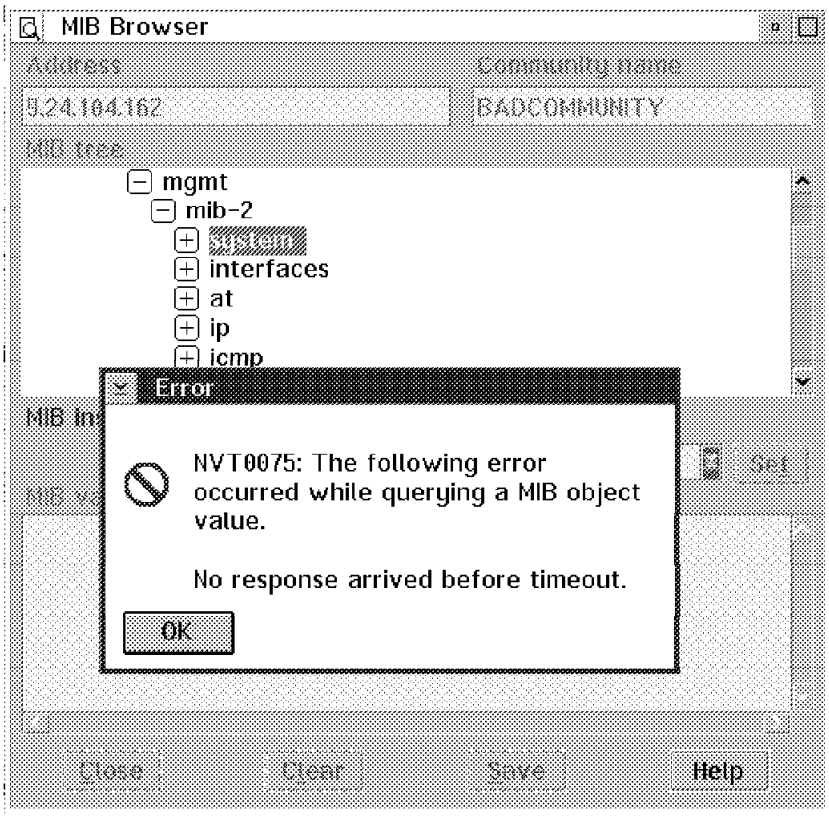


Figure 146. The Wrong Community Name Used

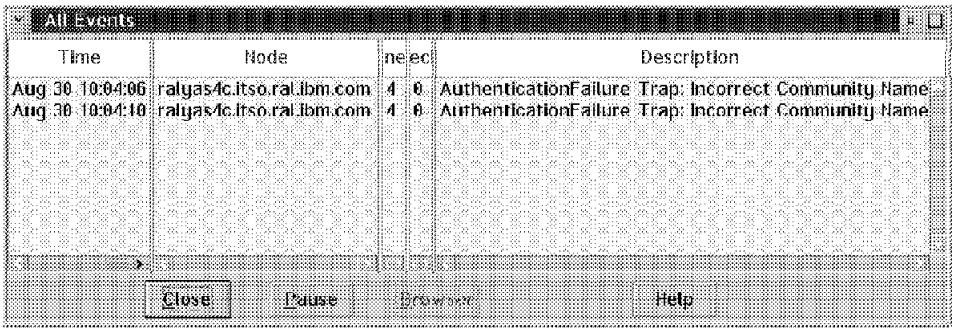


Figure 147. The AuthenticationFailure Trap

In Figure 147 we can see the generic trap number 4 indicating an authentication trap.

Chapter 12. Journal for SNMP Logging

All five SNMP operations (GET, GETNEXT, SET, RESPONSE and TRAP) can be logged in the QSNMP journal for audit purposes.

12.1 QSNMP Journal Overview

A journal is provided with OS/400 SNMP in which all five of the operations (GET, GETNEXT, SET, RESPONSE and TRAP) used to implement the request/response protocol can be logged. To have the SNMP operations logged, you select logging in the SNMP community or in the SNMP attributes. Refer to Chapter 4, “Getting Started” on page 31 for information on how to accomplish this.

The logged records are kept in the journal QSNMP in library QUSRSYS. The DSPJRN (QSNMP/QUSRSYS) command can be used to display the content of the journal. Each record contains 118 columns of alphanumeric data. An example of a journal entry using the DSPJRN (QUSRSYS/QSNMP) command is shown in Figure 148.

```
                            Display Journal Entry

Object . . . . . :                  Library  . . . . . :
Member . . . . . :                  Sequence . . . . . :   10895
Code . . . . . :   M  - Network management data
Type . . . . . :   SN - SNMP information

            Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00001      'TITSC                  9 241041860   04   0       '
00051      '                                                  '
00101      '                  '

                                                            Bottom

Press Enter to continue.

F3=Exit   F6=Display only entry specific data
F10=Display only entry details   F12=Cancel   F24=More keys
```

Figure 148. SNMP Journal Entry Example

The format of the journal entries is as follows:

Column 1: Log Type (type of operation)

G = GET request

N = GETNEXT request

S = SET request

R = RESPONSE

T = TRAP

Column 2-21: Community Name

Column 22-33: Internet Address (IP address of the SNMP manager sending request or receiving response, or the IP address of the first SNMP manager receiving a trap).

Column 34: Error Status (specifies the type of error received).

0 = noError
1 = tooBIG
2 = noSuchName
3 = badValue
5 = genErr

The errors have the following meanings:

tooBig This indicates that the response to a request would be too large to send back in a single packet.

noSuchName This indicates that a request (SET, GET or GETNEXT) contains a variable that is not accessible.

In the case of a SET, a noSuchName error status will be logged when the variable to be modified doesn′t exist or when the variable cannot be modified because it is read only or when the manager is not allowed to modify those agents′ variables.

In the case of a GET or GETNEXT, a noSuchName error status will be logged when the agent does not implement the object associated with the variable or the agent implements the object, but the variable doesn′t exist.

badValue This indicates that the SET operation contains a value that the agent doesn′t like because it may be syntactically inconsistent with the object definition in the MIB or with the values of other variables in the agent.

genError This is a catch-all error which should only be logged when there is no other recourse available.

Column 35-38: Error Index

Column 39: Trap Type (indicates the type of trap received by an SNMP manager).

0: coldStart

1: warmStart

2: linkDown

3: linkUp

4: authenticationFailure

5: egpNeighborLoss

6: enterpriseSpecific

Note: The different type of traps are explained in A.3.2.5, “The Trap-PDU” on page 243.

Column 40-43: Enterprise Specific Trap. If the Trap Type field has a value of 6 (enterpriseSpecific) this field will indicate the type of enterprise type. If the trap type field has any other value, then the value of this field will be 0.

Note: If the enterprise specific trap type value is less than -999, it will be logged as -999. If the enterprise specific trap value is greater than 9999, it will be logged as 9999.

Column 44-118: Descriptors (specifies the name of the MIB variable).

12.2 Examples of Journals Entries

The contents of the journal entries can be displayed using the CL command DSPJRN JRN(QUSRSYS/QSNMP). As shown in in the following examples:

Example 1:

```
                          Display Journal Entry

Object . . . . . :                  Library  . . . . . :
Member . . . . . :                  Sequence . . . . . :   5811
Code . . . . . :   M  - Network management data
Type . . . . . :   SN - SNMP information

           Entry specific data
Column     *...+....1....+....2....+....3....+....4....+....5
00001     'Gpublic               9 18  1 200   00   0sysDesc'
00051     'r.0                                               '
00101     '                 '
```

Figure 149. GET Request

This entry indicates that a GET request specifying community name *public* was sent by an SNMP manager at internet address 9.18.1.20. The error status, error index and the trap type fields all have a value of zero. The object descriptor *sysDescr* is the one that is being requested.

```
                          Display Journal Entry

Object . . . . . :                  Library  . . . . . :
Member . . . . . :                  Sequence . . . . . :   5812
Code . . . . . :   M  - Network management data
Type . . . . . :   SN - SNMP information

           Entry specific data
Column     *...+....1....+....2....+....3....+....4....+....5
00001     'Rpublic               9 18  1 200   00   0sysDesc'
00051     'r.0                                               '
00101     '                 '
```

Figure 150. Response to GET Request

This journal entry is the log of the response to the previous GET request. The only field of this entry that has changed is the log type. The error status and error index field values are zero, so no error was registered in this GET operation. The field object descriptor indicates the object returned in response to the GET request.

Example 2:

```
                          Display Journal Entry

Object . . . . . :                  Library  . . . . . :
Member . . . . . :                  Sequence . . . . . :   5813
Code . . . . . :   M  - Network management data
Type . . . . . :   SN - SNMP information

            Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00001      'Gpublic               9 18  1 200   00   0sysDesc'
00051      'r                                                 '
00101      '                 '
```

Figure 151. Incorrect Object ID being Specified in a GET Request

This entry indicates that a GET request specifying community *public* was sent by an SNMP manager at internet address 9.18.1.20. The error status, error index and the trap type fields all have a value of zero. The object descriptor sysDescr is the one that is being requested, but it was not correctly specified (it should have been sysDescr.0)

```
                          Display Journal Entry

Object . . . . . :                  Library  . . . . . :
Member . . . . . :                  Sequence . . . . . :   5814
Code . . . . . :   M  - Network management data
Type . . . . . :   SN - SNMP information

            Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00001      'Rpublic               9 18  1 202   10   0sysDesc'
00051      'r                                                 '
00101      '                 '
```

Figure 152. Response to an Incorrect GET Request

This journal entry is the log of the response to the previous GET request. The error status and error index fields show that the *noSuchName* error was returned for the object specified in the object descriptor field.

Example 3:

```
                          Display Journal Entry

Object . . . . . :                  Library  . . . . . :
Member . . . . . :                  Sequence . . . . . :   5465
Code . . . . . :   M  - Network management data
Type . . . . . :   SN - SNMP information

            Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00001      'Npublic               9 241041860   00   OsysCont'
00051      'act                                              '
00101      '                 '
```

Figure 153. GETNEXT Request

This journal entry indicates that a GETNEXT request specifying the community name *public* was sent by an SNMP manager at internet address 9.24.104.186. Error status, error index, and the trap type fields all have the value zero. In this case we are requesting the content of the next MIB variable of object descriptor *sysContact*.

```
                          Display Journal Entry

Object . . . . . :                  Library  . . . . . :
Member . . . . . :                  Sequence . . . . . :   5466
Code . . . . . :   M  - Network management data
Type . . . . . :   SN - SNMP information

            Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00001      'Rpublic               9 241041860   00   OsysCont'
00051      'act.0                                            '
00101      '                 '
```

Figure 154. Response to GETNEXT Request

This journal entry is the log of the response to the previous GETNEXT request. The only fields of this entry that have changed are the log type and the object descriptor. The field object descriptor indicates the object returned in response to the GETNEXT request.

Example 4:

```
                          Display Journal Entry

Object . . . . . :                  Library  . . . . . :
Member . . . . . :                  Sequence . . . . . :   5467
Code . . . . . :   M  - Network management data
Type . . . . . :   SN - SNMP information

            Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00001      'Spublic               9 241041860   00   0sysCont'
00051      'act.0                                             '
00101      '                 '
```

Figure 155. SET Request

This entry indicates that a SET request specifying community name *public* was sent by an SNMP manager at internet address 9.24.104.186. The error status, error index and the trap type fields all have a value of zero. The object descriptor *sysContact* is the one that is requested to be modified by the SET operation.

```
                          Display Journal Entry

Object . . . . . :                  Library  . . . . . :
Member . . . . . :                  Sequence . . . . . :   5468
Code . . . . . :   M  - Network management data
Type . . . . . :   SN - SNMP information

            Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00001      'Rpublic               9 241041860   00   0sysCont'
00051      'act.0                                             '
00101      '                 '
```

Figure 156. Response to SET Request

This journal entry is the log of the response to the previous SET request. The only field of this entry that has changed is the log type. The field object descriptor indicates the object changed in response to the SET request.

Example 5:

```
                          Display Journal Entry

Object . . . . . :                  Library  . . . . . :
Member . . . . . :                  Sequence . . . . . :   5929
Code . . . . . :   M  - Network management data
Type . . . . . :   SN - SNMP information

            Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00001      'Spublic               9 18  1 200   00   0sysUpTi'
00051      'me.0                                              '
00101      '                 '
```

Figure 157. Modifying a Read-Only Object ID

This entry indicates that a SET request specifying community name *public* was sent by an SNMP manager at internet address 9.18.1.20. The error status, error index and the trap type fields all have a value of zero. The object descriptor *sysUpTime* is the one that is requested to be modified by the SET operation.

```
                          Display Journal Entry

Object . . . . . :                  Library  . . . . . :
Member . . . . . :                  Sequence . . . . . :   5930
Code . . . . . :   M  - Network management data
Type . . . . . :   SN - SNMP information

            Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00001      'Rpublic               9 18  1 202   10   0sysUpTi'
00051      'me.0                                              '
00101      '                 '
```

Figure 158. Response to an Incorrect SET Request

This journal entry is the log of the response to the previous SET request. The error status and error index fields show that the *noSuchName* error was returned for the object specified in the object descriptor field. In this case *noSuchName* was returned because the object descriptor field *sysUpTime* is read only so it cannot be modified.

Example 6:

```
                           Display Journal Entry

Object . . . . . :                  Library  . . . . . :
Member . . . . . :                  Sequence . . . . . :   5443
Code . . . . . :   M  - Network management data
Type . . . . . :   SN - SNMP information

            Entry specific data
Column      *...+....1....+....2....+....3....+....4....+....5
00001      'TLEE                   9 241041860   04   0       '
00051      '                                                  '
00101      '                  '
```

Figure 159. authenticationFailure Trap

This journal entry indicates that a trap was sent to the trap manager at internet address 9.24.104.186. The trap will be sent to a trap manager only if the trap manager job is started. For more information, refer to 6.1.1, “SNMP Manager API Functions” on page 76. Error status and error index are both zero, the trap type is authenticationFailure. In this case the wrong community name was specified causing the *authenticationFailure* trap to be generated.

Chapter 13. Loading an Enterprise-Specific MIB

Internet standard MIBs are supported by all devices that support SNMP. The standard MIB objects definition, MIB-2, enables you to monitor and control SNMP devices.

Vendors can also define enterprise-specific MIBs for controlling their own devices. By loading a MIB description file containing one of these enterprise-specific MIBs on to an SNMP manager, you can control these devices.

13.1 OS/400 and IBM Enterprise MIBs

IBM enterprise MIB modules supported by OS/400 are shipped with OS/400. Each MIB module is contained in a member of the physical file QANMMIB in library QSYS. See Figure 160.

```
                          Work with Members Using PDM                RALYAS4B

File  . . . . . .   QANMMIB
  Library . . . .     QSYS                 Position to  . . . . .

Type options, press Enter.
 2=Edit         3=Copy  4=Delete 5=Display       6=Print     7=Rename
 8=Display description  9=Save  13=Change text  14=Compile  15=Create module

Opt  Member      Type        Text
 5   IBMAPPN
     IBMCLTM
     IBMNV6SA
     IBMRWS    1
     IBMSNPSA  1



Parameters or command
===>                                                                  Bottom
F3=Exit          F4=Prompt             F5=Refresh            F6=Create
F9=Retrieve      F10=Command entry     F23=More options      F24=More keys
```

Figure 160. OS/400 Supplied MIB Modules in File QSYS/QANMMIB

If we select option 5 (display) against member IBMAPPN, the MIB module source code shown in Figure 161 on page 220 will be displayed.

Note: The MIB source file members marked by **1** are not available in V3R1 or V3R2.

```
 SEU==>
 FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..
        *************** Beginning of data ***************data *******************
0001.00
0002.00 -- Note:
0003.00 -- This text file contains ASN.1 source describing MIB objects for SNMP.
0004.00 -- The enterprise MIB described is subject to change in a future
0005.00 -- release, and support for any object described in this MIB may be
0006.00 -- removed in a future release, as standard MIBs in the management domain
0007.00 -- described by this MIB become defined and published as Internet RFCs.
0008.00
0009.00 IBM-6611-APPN-MIB DEFINITIONS  ::= BEGIN
0010.00
0011.00 IMPORTS
0012.00
0013.00         enterprises, Counter, IpAddress,
0014.00         Gauge, TimeTicks
0015.00                FROM RFC1155-SMI
0016.00

 F19=Left   F20=Right   F21=System command             command
 F24=More keys
```

Figure 161. IBMAPPN MIB Source Code

This chapter explains how to load one of these enterprise MIB modules on to an SNMP manager (NetView for OS/2 in this example). We will then use it to browse information on the AS/400.

The procedures and commands used in the following examples are specific to NetView for OS/2. Other platforms will almost certainly use slightly different procedures.

13.2 Transferring the MIB Module to the Manager

To transfer the MIB module to the PC running NetView for OS/2 we will be using the TCP/IP File Transfer Protocol (FTP). In this example we will be transferring the MIB module IBMAPPN.

At present we can use the MIB Browser to go down the MIB tree to the MIB object ibmappn. However, it is not possible to go any further down the tree into the more granular objects and variables without the framework that the MIB will provide for the manager.

Note

The MIB object ibmappn is reached via the following route:

1. private
2. enterprises
3. ibm
4. ibmProd
5. ibm6611
6. ibmappn

Clicking on the **ibmappn** MIB object will not reveal any lower level MIB objects unless the ibmappn enterprise MIB is loaded.

13.2.1 Transferring a MIB Module Using FTP

Open an OS/2 window on your PC, then enter the following command:

```
ftp IP_address
```

Here the IP_address is the internet address of the AS/400 (in this example it is 9.24.104.57). See Figure 162.

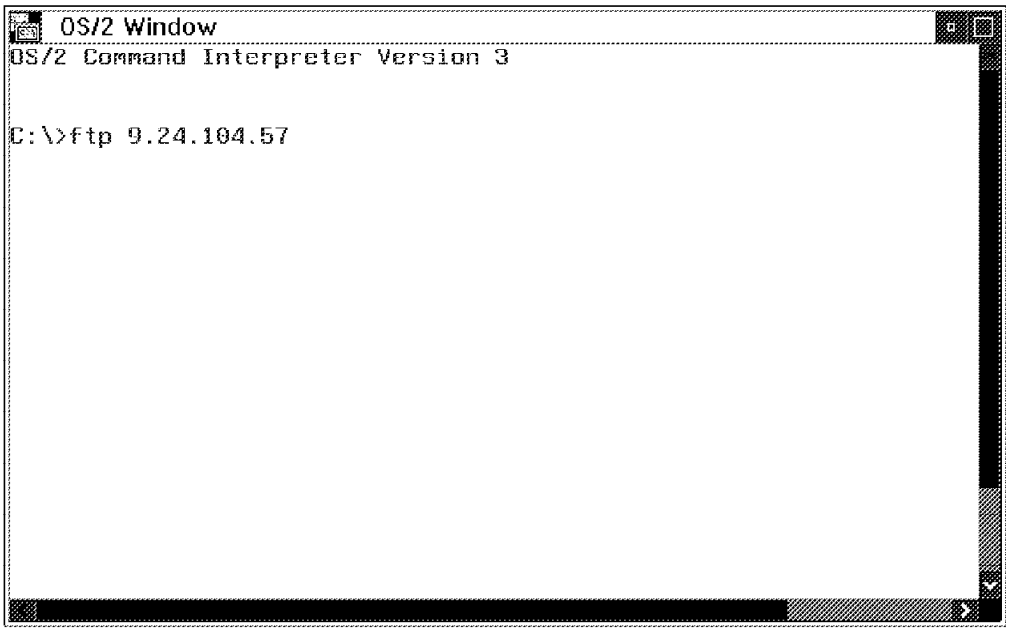


Figure 162. Connecting to the AS/400 Using FTP

Enter your AS/400 user ID and password at the prompt. You should then receive confirmation that you have started an FTP connection and be presented with an FTP command prompt. At the FTP command prompt enter the following command:

```
cd QSYS
```

This should change your current library to QSYS on the AS/400. See Figure 163 on page 222.

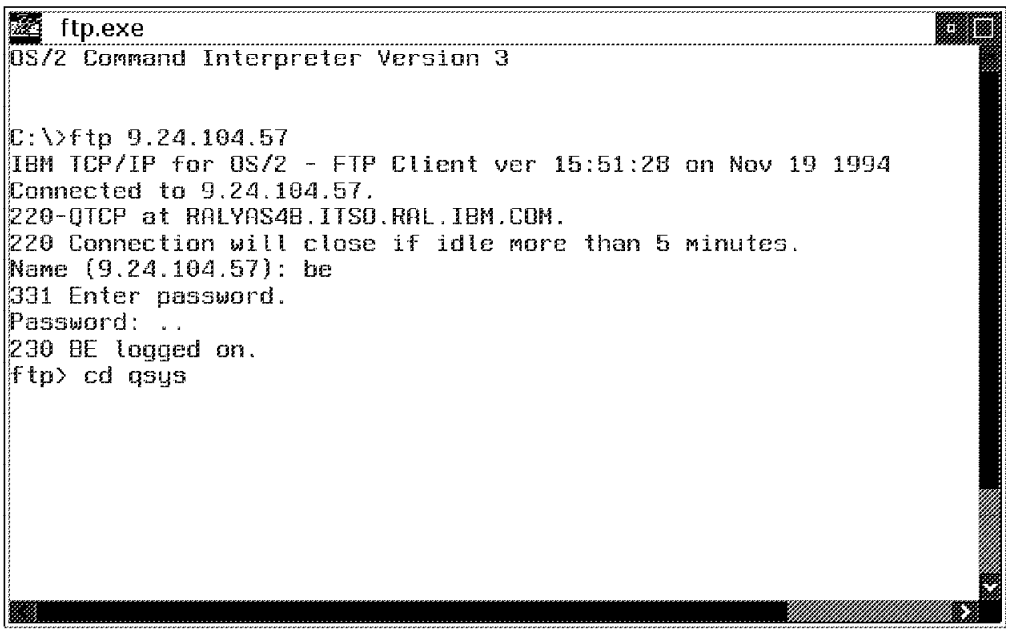


Figure 163. Change Current Library to QSYS on the AS/400

Next, enter the following command:

```
get qanmmib.ibmappn d:\anv2\snmp_mib\ibmappn
```

See Figure 164.

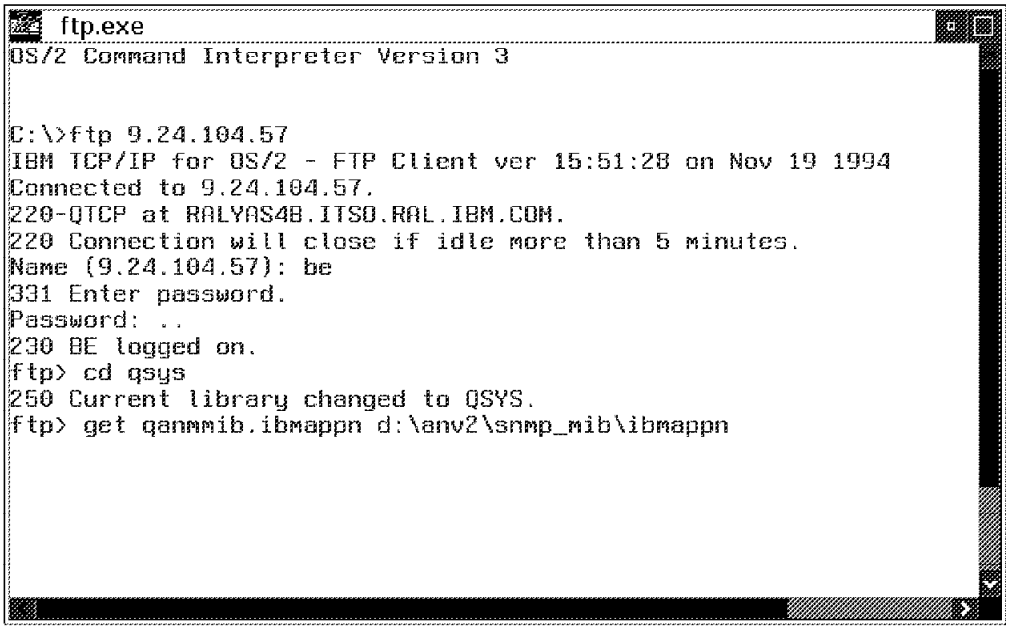


Figure 164. Transferring the MIB Using the FTP GET Command

This should copy the AS/400 member IBMAPPN from file QANMMIB in library QSYS to directory SNMP_MIB on the PC. Check that this directory is reached via the same path on your PC.

You should receive a message that the data has been transferred successfully. See Figure 165 on page 223.

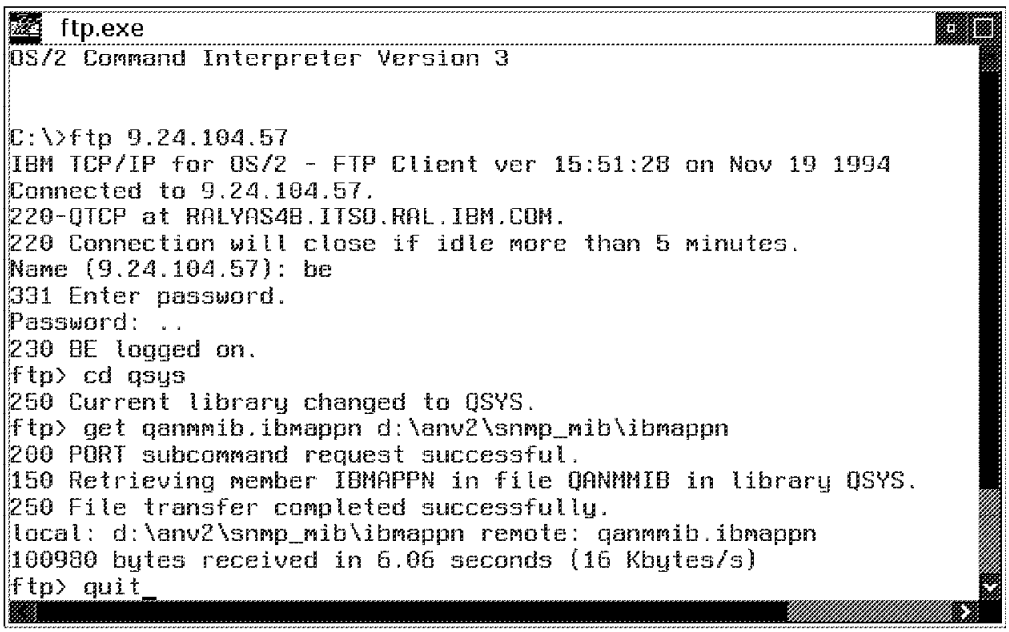


Figure 165. MIB Successfully Transferred

You can use a PC text editor to check if the file you received is the same as the AS/400 file member you displayed earlier in Figure 161 on page 220.

Figure 166 shows the file we received on our PC viewed using a PC text editor.

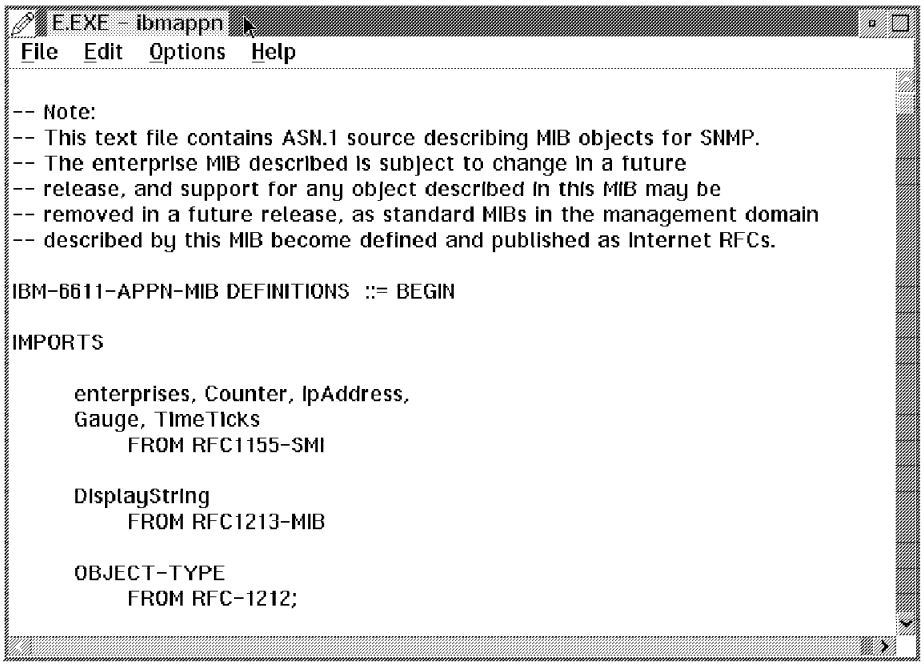


Figure 166. Viewing the Transferred MIB Source Code

13.2.2 Loading the MIB into NetView for OS/2

The received MIB module must now be loaded into NetView for OS/2 using the LOADMIB command. Enter the following command from the OS/2 prompt:

```
loadmib -load ibmappn
```

See Figure 167 for an example.

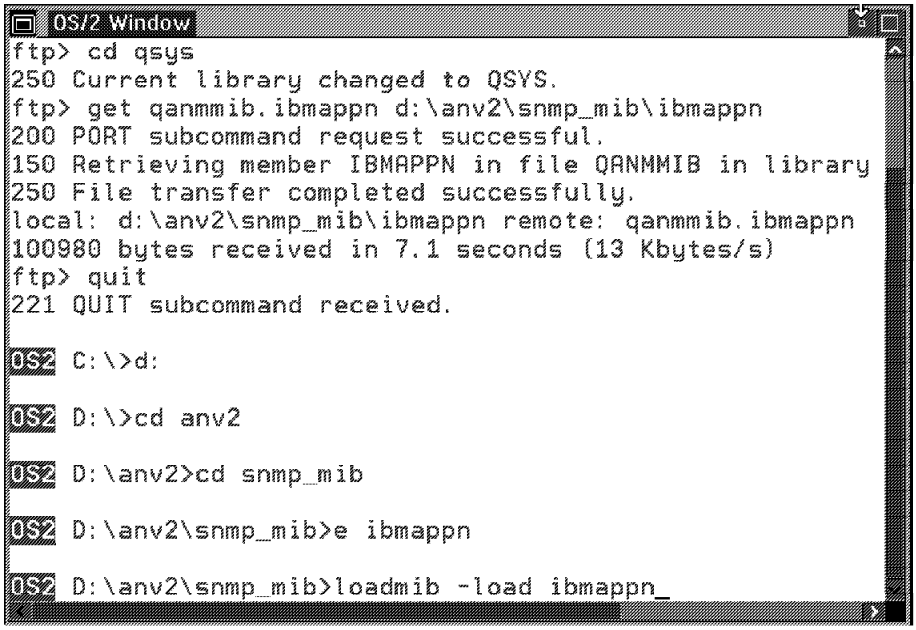


Figure 167. Loading the MIB into NetView for OS/2

You should then see the loadmib procedure running. When you receive the message Command completed successfully the MIB should be loaded. See Figure 168.

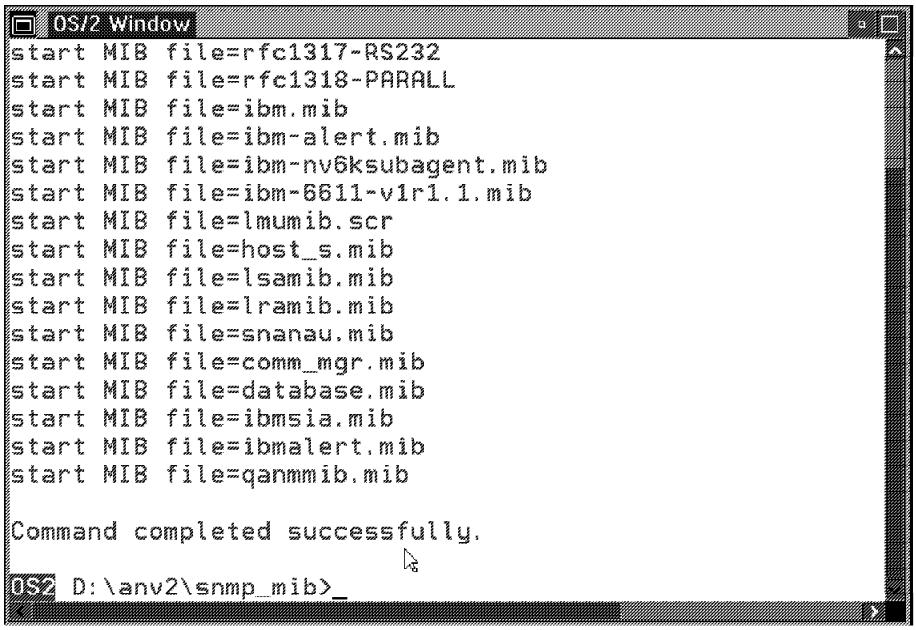


Figure 168. LOADMIB Command Completed Successfully

You should now be able to use the MIB Browser to browse the objects in the *ibmappn* group and retrieve data from the MIB variables. See Figure 169 on page 225 which shows the variable *ibmappnLocalNodeType* (APPN local node type), in this case a network node.

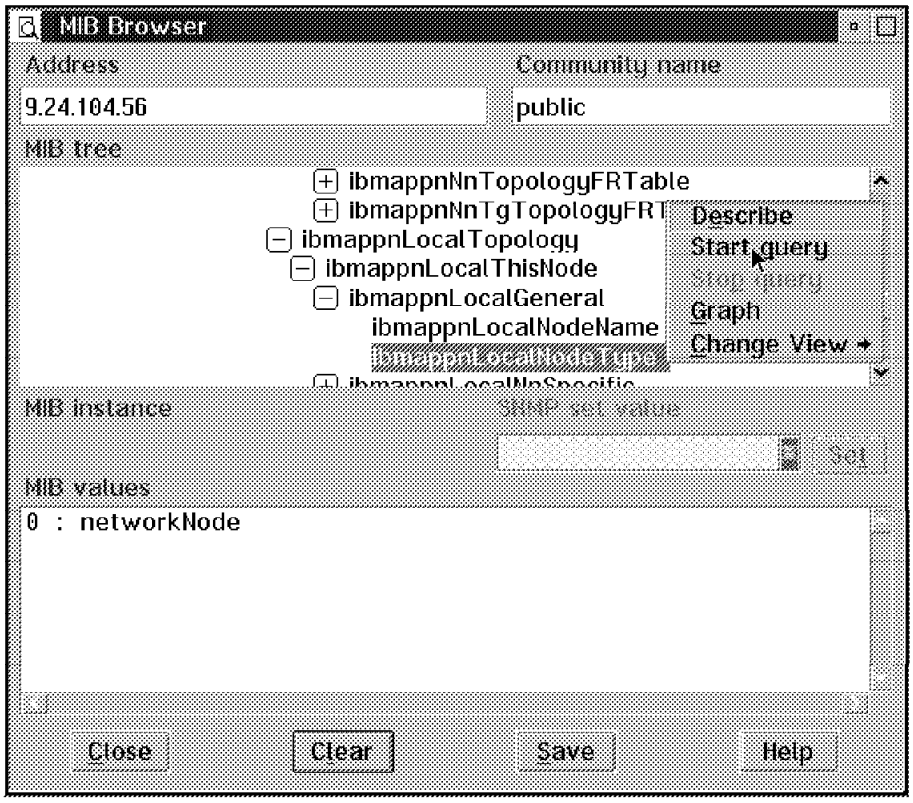


Figure 169. Browsing ibmappn MIB Variables

We used this MIB in 9.4.1, “Advanced Peer-To-Peer Networking (APPN) MIB” on page 159.

Chapter 14. SNMP and AnyNet

In this chapter we discuss AnyNet and how SNMP can utilize the functions of AnyNet.

14.1 AnyNet Overview

AnyNet is a family of products designed to make applications independent from a specific transport protocol. AnyNet products implement the Multiprotocol Transport Networking (MPTN) architecture. AS/400 AnyNet support was added to OS/400 at V3R1 and allows the use of:

- APPC over TCP/IP
- Sockets over SNA
- APPC over IPX
- Sockets over IPX

An example of a solution provided by AnyNet is the following:

A customer may have implemented a large and costly SNA network but would like to use the TCP/IP File Transfer Protocol (FTP) to transfer files between some systems. AnyNet allows them to use FTP across their existing network without the need to implement a parallel TCP/IP network.

14.1.1 APPC over TCP/IP

APPC over TCP/IP allows APPC applications to communicate over TCP/IP networks. LU 6.2 APPC or CPI-C applications can be added to an existing TCP/IP network.

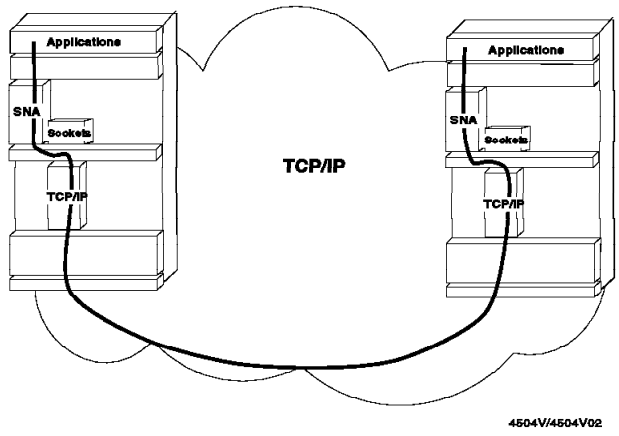


Figure 170. APPC over TCP/IP

14.1.2 Sockets over SNA

Sockets over SNA allows sockets applications to communicate over SNA networks. Applications written to the socket interface can be added to an existing SNA network. Some of the socket applications that are supported under AnyNet/400 are as follows:

- Simple Network Management Protocol (SNMP)
- File Transfer Protocol (FTP)
- Remote Printing (LPR and LPD)

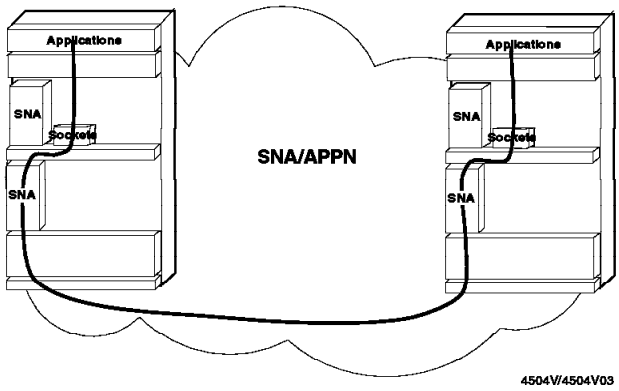


Figure 171. Sockets over SNA

14.1.3 Multiprotocol Transport Networking (MPTN) Architecture

The MPTN architecture is defined in the terminology of the IBM Networking Blueprint. AnyNet products implement the MPTN architecture. In Figure 172 on page 229 the arrows depict the way the MPTN architecture, by delivering CTS (Common Transport Semantics) function, allows applications designed to run over one transport network to run over another. The arrows depict APPC applications over TCP/IP and sockets applications over SNA.

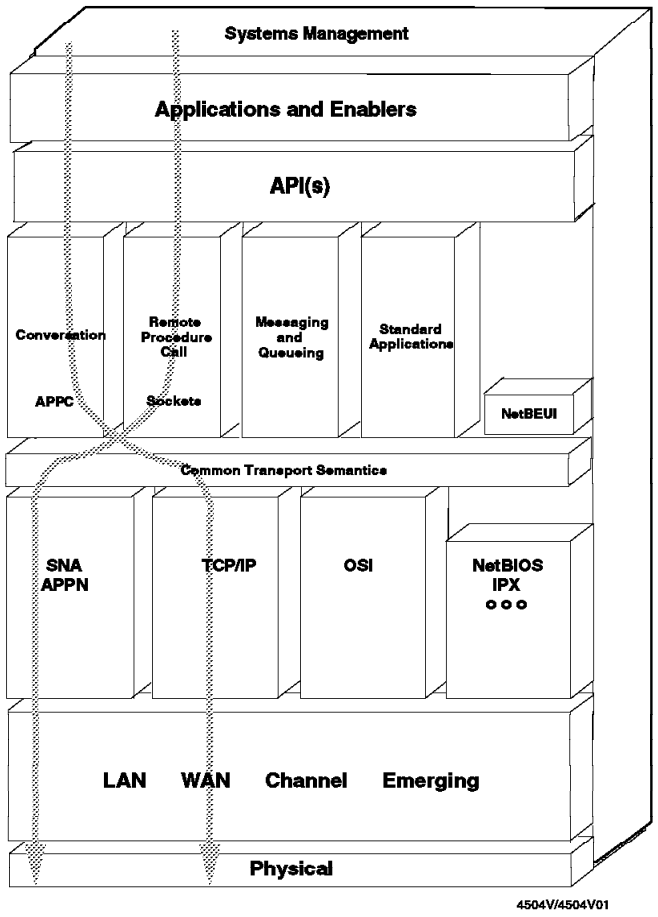


Figure 172. MPTN Architecture Model

14.1.4 AnyNet/400 Considerations

Although AnyNet/400 can, in some cases, provide a performance improvement for TCP/IP applications running over SNA; the extra processing involved usually results in a slight performance degradation. You should note that any sockets applications running natively over TCP/IP will suffer a performance degradation if your system implements AnyNet/400.

These are points to consider when you decide whether to implement AnyNet/400 support. Normally, however, the flexibility of the AnyNet/400 product should outweigh any performance degradation.

14.2 SNMP AnyNet Scenarios

In Figure 173, the Client Access/400 PC is attached to the AS/400 A via a TCP/IP network. Client Access/400, being an APPC application, is using APPC over TCP/IP to communicate with the AS/400. SNMP can be used natively (over TCP/IP) between the Client Access/400 PC and the AS/400. Also shown is an AS/400 connected to a NetView for OS/2 system via an SNA network. In this case Sockets over SNA allows SNMP to be used between the NetView for OS/2 system and the AS/400 B via the SNA network.

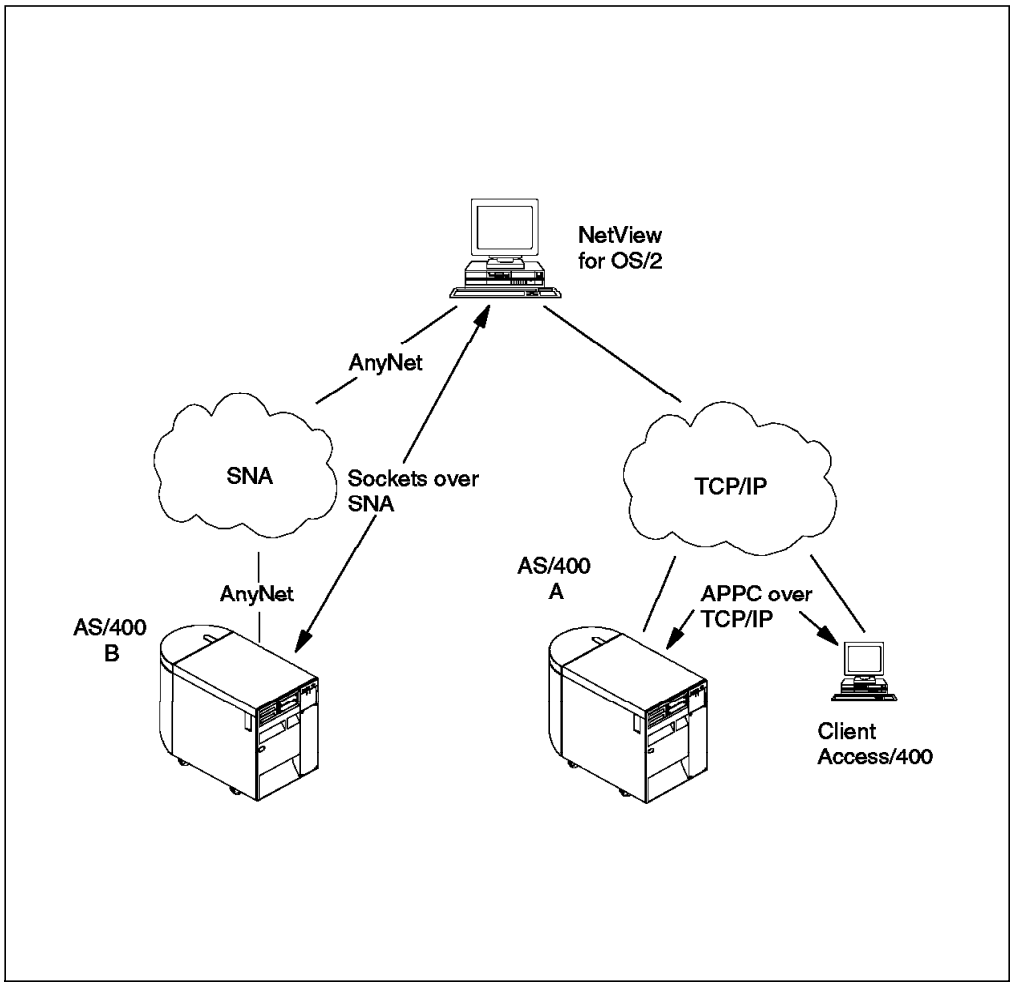


Figure 173. SNMP over Native TCP/IP and AnyNet

Chapter 15. NetView for AIX

In this book we have tended to concentrate on NetView for OS/2 as the SNMP manager. In this chapter we take a short look at another SNMP manager, NetView for AIX.

15.1 Overview

NetView for AIX Version 1 was first introduced during 1992. It was 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.

The most noticeable difference between NetView for AIX and NetView for OS/2 is the former's graphical user interface. Using information gleaned from SNMP agents, NetView for AIX can construct network topology maps allowing network administrators to see, at a glance, the current state of systems on the network.

Most of the features of NetView for OS/2 are present on NetView for AIX such as, the discovery process, event automation and the MIB browser. The discovery process, however, when coupled to the graphical user interface, is more sophisticated.

The NetView for AIX discovery process can do the following:

- Discover a new entity on the network
- Discover new devices from a device already discovered
- Poll nodes for any status change that may have occurred
- Poll nodes for any configuration changes
- Determine the type of system or node discovered by using the sysDescr object in MIB II

The process that deals with discovery is called the netmon daemon. It will discover the following 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 it will discover these entities, it will initially place them in an *unmanaged* state.

15.2 NetView for AIX Graphical Interface

As systems and networks are discovered, a pictorial view or map can be drawn to represent them.

See Figure 174 on page 232 for a graphical representation of a network as discovered and displayed by NetView for AIX.

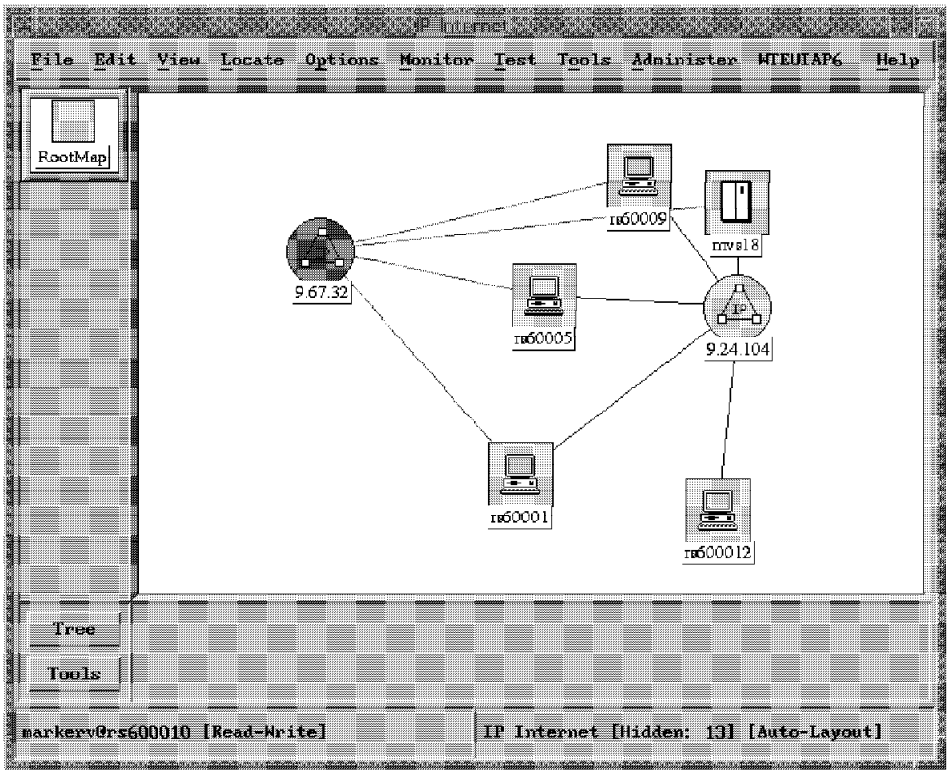


Figure 174. Network Discovered by NetView for AIX

Figure 174 shows the home network, 9.24.104, all the bridges, routers and gateways attached to the home network and networks attached to these devices.

You can see that NetView for AIX has discovered another network , 9.67.32. It displays this in red (not green) because it cannot reach this network as it has no route defined. It discovered the existence of this network by retrieving interface information from the systems attached to it. By using SNMP, NetView for AIX found that these systems had an interface into the 9.24.104 network and also the 9.67.32 network.

For a more meaningful view of the network it is possible to superimpose the network on a user-defined background. Figure 175 on page 233 shows our network superimposed on a map of the USA. It is now possible to place the device icons in their actual locations.

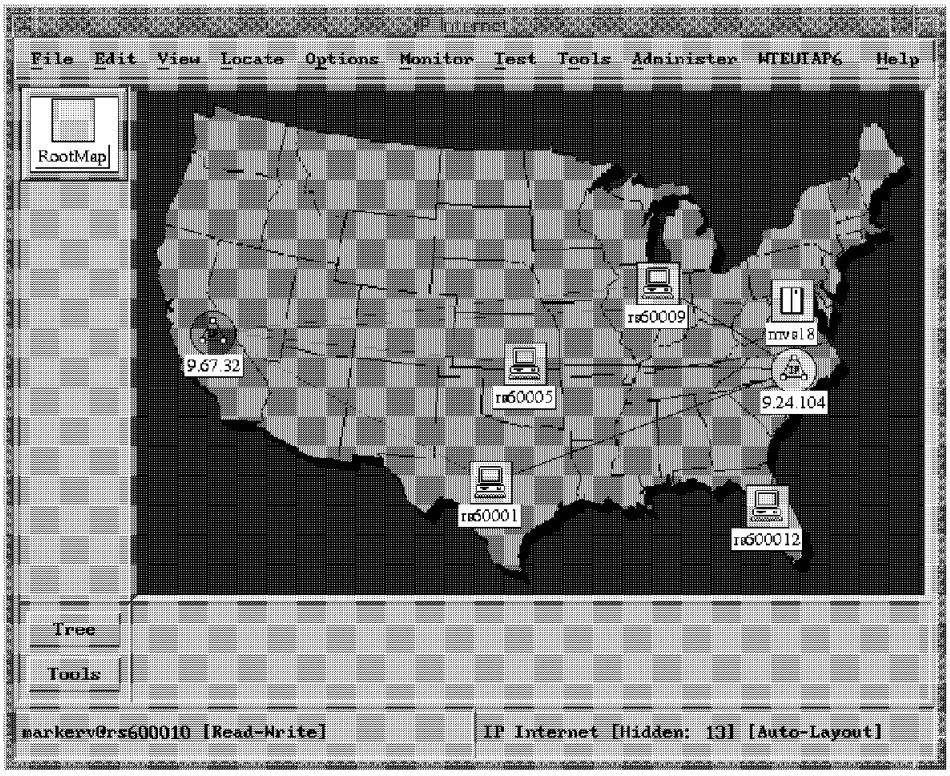


Figure 175. Network Superimposed on a Meaningful Background

It is possible to examine the network in closer detail by *zooming in* on a network, see Figure 176.

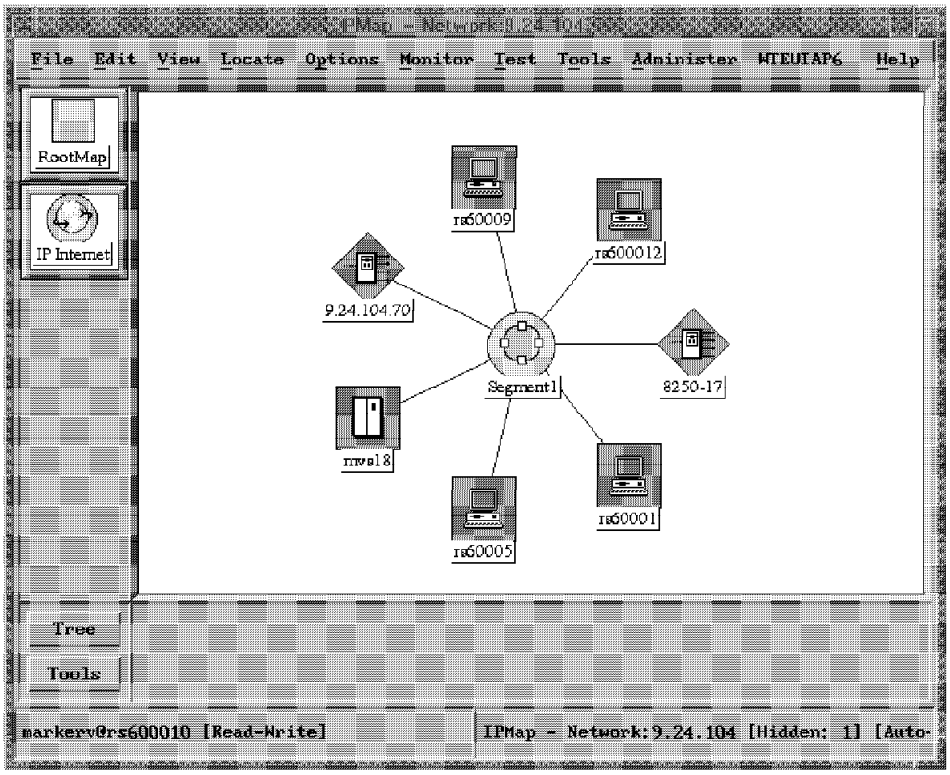


Figure 176. A LAN Segment Viewed from NetView for AIX

Figure 176 shows just our home LAN segment and the bridges, routers and gateways attached to it. To see all the systems on our LAN segment, we can look at a different view. Figure 177 on page 234 shows all the systems attached to our segment.

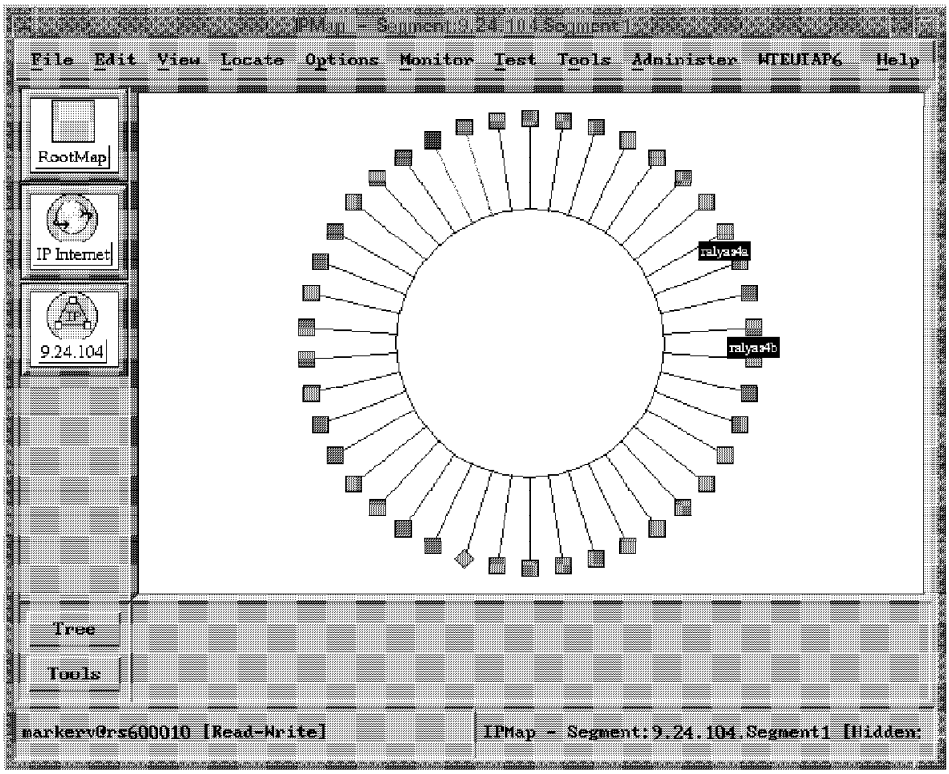


Figure 177. All Systems on our LAN Segment

You can now see systems RALYAS4A and RALYAS4B on this segment. Because the graphical display uses a different color to represent each different status of a system (such as communicating or not communicating) it is very easy to tell whether a system is online or not. Should any system fail, the status would turn from green to red. If a gateway failed, many systems on this map may turn red. It would be a simple matter of viewing the entire network map to see which gateway, or router, had failed. If you are using a pictorial view of the network, as shown in Figure 175 on page 233, you would even know which location the failed device was in and which parts of the country would be affected.

AS/400 systems with an active SNMP agent will be discovered by NetView for AIX. NetView for AIX Version 3 will recognize the AS/400 sysDescr object and show the object type as IBM AS/400.

Appendix A. Additional Information on SNMP

This appendix contains information for further reading relative to SNMP. Some topics from the preceding chapters are explored in more detail, as well as other topics relevant to SNMP and the Internet protocol suite in general.

A.1 The MIB Tree Structure

As mentioned previously in 2.9.2, “MIB Naming Conventions” on page 16, the MIB tree structure originates from a root. The root of the tree itself is unlabeled, but has at least three children directly under it:

- International Organization for Standardization (ISO) node with the label iso(1).
- International Telegraph and Telephone Consultative Committee (CCITT) node with the label ccitt(0).
- A node jointly administered by ISO and CCITT with the label joint-iso-ccitt(2).

Under the iso(1) node, the ISO has designated one subtree for use by other international organizations, identified with label org(3). Of the two children nodes present, two have been assigned to the U.S. National Institute of Standards and Technology (NIST). One of these subtrees has been transferred by the NIST to the U.S. Department of Defense (DoD), which is identified by the label dod(6). To date, the DoD has not indicated how it will manage its subtree of OBJECT IDENTIFIERs. The IAB assumes that the DoD will allocate a node to the Internet community to be administered by the IAB as follows:

```
internet      OBJECT IDENTIFIER ::= iso org(3) dod(6) 1
```

That is, the Internet subtree of OBJECT IDENTIFIERs starts with the prefix 1.3.6.1.

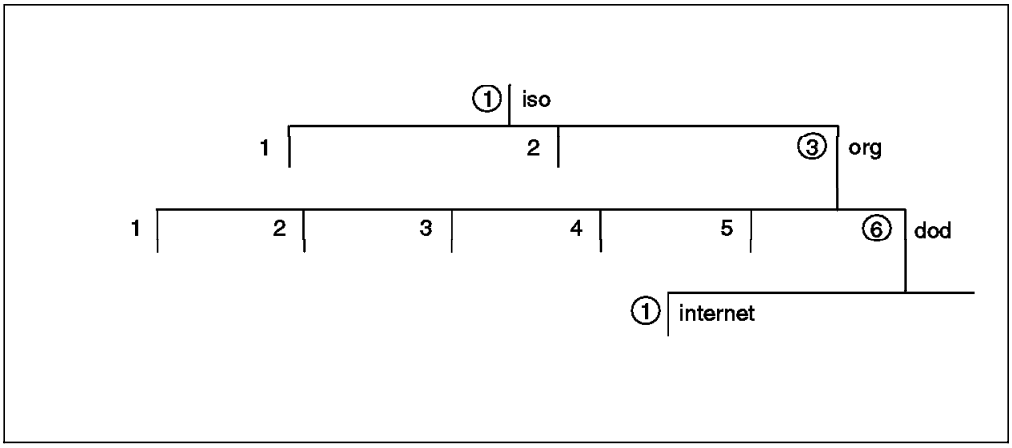


Figure 178. A View of the Internet Subtree

The SMI standard RFC specifies that the Internet subtree of OBJECT IDENTIFIERs will contain the following four nodes:

- directory OBJECT IDENTIFIER ::= internet 1
- mgmt OBJECT IDENTIFIER ::= internet 2
- experimental OBJECT IDENTIFIER ::= internet 3

- private OBJECT IDENTIFIER ::= internet 4

A.1.1 Directory Subtree

The directory(1) subtree is reserved for use with a future RFC that discusses how the OSI Directory may be used in the Internet.

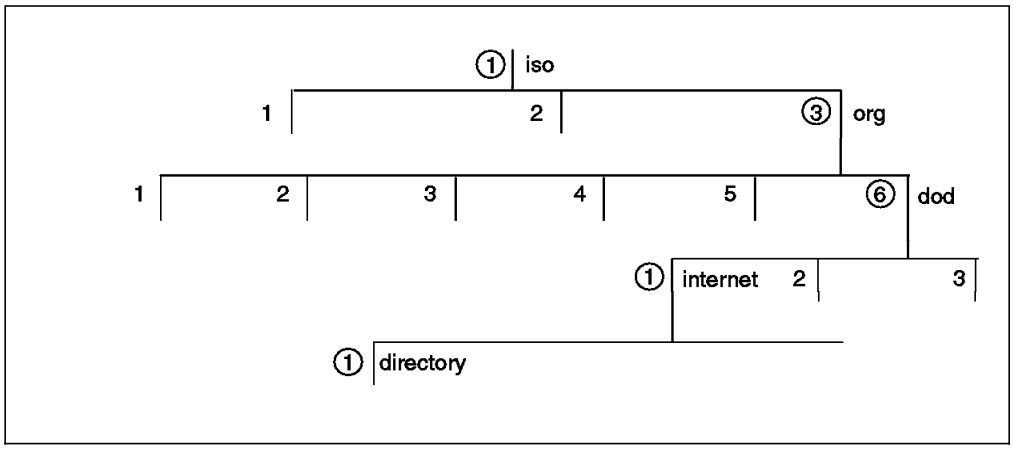


Figure 179. A View of the Directory Subtree

A.1.2 Mgmt Subtree

The mgmt(2) subtree is used to identify objects which are defined in IAB-approved documents. Administration of the mgmt(2) subtree is delegated by the IAB to the *Internet Assigned Numbers Authority (IANA)*. As RFCs which define new versions of the Internet standard MIB are approved, they are assigned an OBJECT IDENTIFIER by the IANA for identifying the objects defined by that RFC. For example, the RFC which defines the initial Internet standard MIB would be assigned management document number 1. To define the Internet standard MIB, this RFC would use the OBJECT IDENTIFIER:

mgmt 1 or 1.3.6.1.2.1

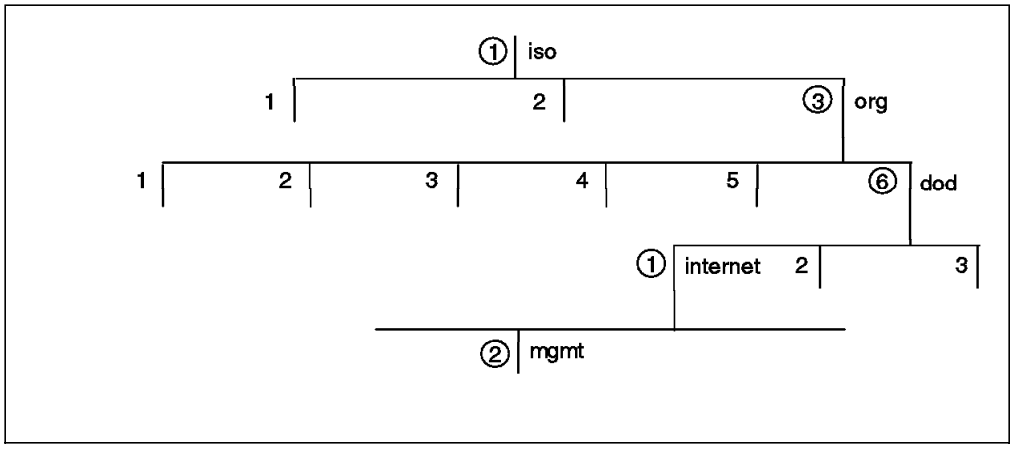


Figure 180. A View of the Mgmt Subtree

A.1.3 Experimental Subtree

The experimental(3) subtree is used to identify objects used in Internet experiments. The IANA administers this subtree. For example, an experimenter might receive number 24, and would have available for use, the OBJECT IDENTIFIER:

experimental 24 or 1.3.6.1.3.24

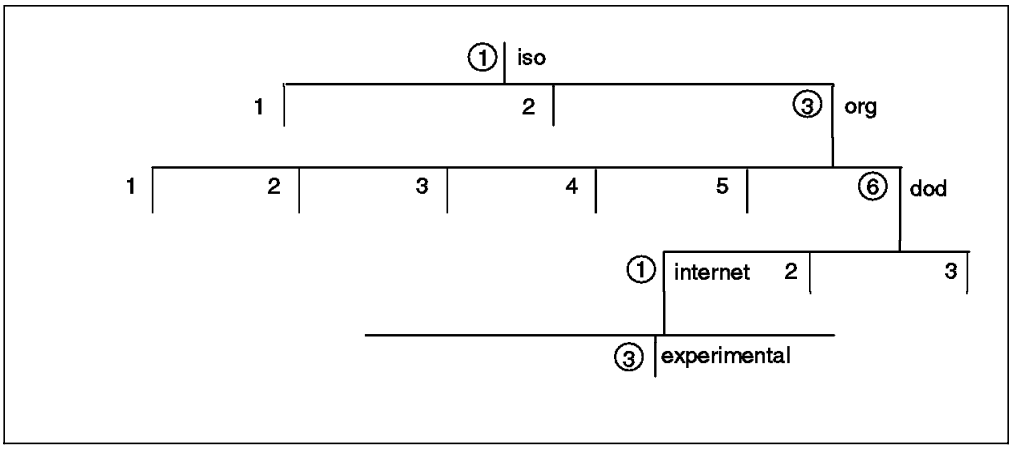


Figure 181. A View of the Experimental Subtree

A.1.4 Private Subtree

The private(4) subtree is used to identify objects defined for private use. The IANA administers this subtree. Initially, this subtree has at least one child:

enterprises OBJECT IDENTIFIER ::= private 1

The enterprises(1) subtree is used, among other things, to allow parties providing networking subsystems to register models of their products. That is, when an enterprise receives a subtree, it may define new MIB objects under the subtree, including its networking subsystems, in order to provide an unambiguous identification mechanism for use in management protocols. For example, if the ″99 Flavors, Inc.″ enterprise produced networking subsystems, then it could request a node under the enterprises subtree from the IANA. Such a node might be numbered: 1.3.6.1.4.1.59

The ″99 Flavors, Inc.″ enterprise might then register its ″Vanilla Router″ under the name of: 1.3.6.1.4.1.59.1.1

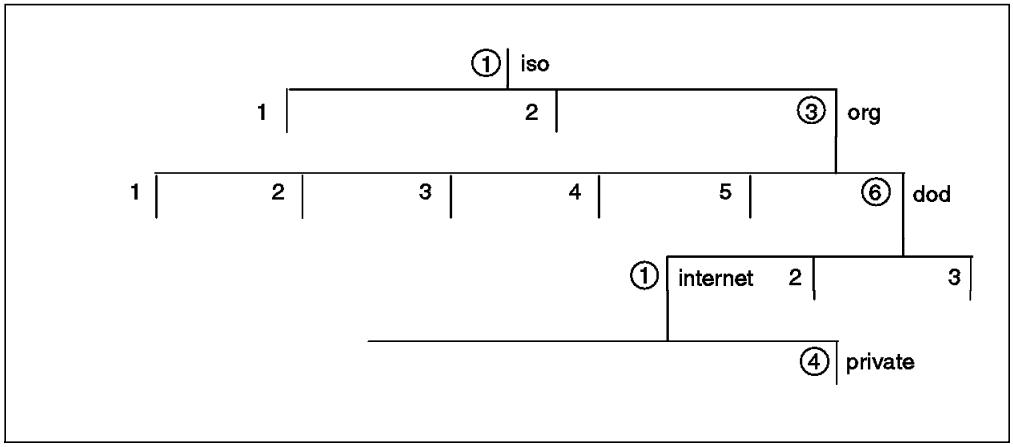


Figure 182. A View of the Private Subtree

A.2 Administrative SNMP Terminology

This section includes some useful SNMP concepts and definitions taken from RFC1157, which refer to the administrative relationships involved in the protocol.

SNMP application entities
These are the entities residing at management stations and network elements which communicate with one another using SNMP.

Protocol entities
These are the peer processes which implement SNMP, and thus support the SNMP application entities.

SNMP community
This is a pairing of an SNMP agent with some arbitrary set of SNMP application entities.

Community name
This is the administratively assigned string of characters which make up the name of an SNMP community.

Authentic SNMP message
This is an SNMP message originated by an SNMP application entity that in fact belongs to the SNMP community named by the community component of said message.

Authentication scheme
This is the set of rules by which an SNMP message is identified as an authentic SNMP message for a particular SNMP community.

Authentication service
This is an implementation of a function that identifies authentic SNMP messages according to one or more authentication schemes. The degree of certainty an authentication service offers is implementation-specific.

Some implementations may support only a trivial authentication service that identifies all SNMP messages as authentic SNMP messages. However, for an effective management of administrative relationships, it is advisable to implement an authentication service that offers a higher degree of certainty.

SNMP MIB view

This is a subset of objects in the MIB that pertains to a given network element. The names of the OBJECT-TYPEs represented in an SNMP MIB view need not belong to a single subtree of that OBJECT-TYPE. The OS/400 SNMP agent does not support SNMP MIB views. Each OS/400 community consists of all the objects in the MIB.

SNMP access mode

This is an element of the set READ-ONLY or READ-WRITE.

SNMP community profile

This is a pairing of an SNMP access mode with an SNMP MIB view. An SNMP community profile represents specified access privileges to variables in a specified MIB view. For every variable in the MIB view in a given SNMP community profile, access to that variable is represented by the profile according to the following conventions:

- If the said variable is defined in the MIB with the access of *none*, it is unavailable as an operand for any operator.
- If the said variable is defined in the MIB with the access of *read-write*, or *write-only*, and the access mode of the given community profile is *READ-WRITE*, that variable is available as an operand for the GET, SET and TRAP operations.
- Otherwise, the variable is available as an operand for the GET, and TRAP operations.
- In those cases where a *write-only* variable is an operand used for the GET or TRAP operations, the value given for the variable is implementation-specific.

SNMP access policy

This is a pairing of an SNMP community with an SNMP community profile. An access policy represents a specified community profile afforded by the SNMP agent of a specified SNMP community to other members of that community. All administrative relationships among SNMP application entities are architecturally defined in terms of SNMP access policies.

SNMP proxy access policy

This is the type of access policy which exists if the network element on which the SNMP agent for the specified SNMP community resides is not that to which the MIB view for the specified profile pertains.

SNMP proxy agent

This is the SNMP agent associated with a proxy access policy.

Variable binding (VarBind)

This is the pairing of the name of a variable to the variable′s value.

VarBindList

This is a simple list of variable names and corresponding values.

A.3 SNMP Messages

A message consists of a version identifier, an SNMP community name, and a Protocol Data Unit (PDU). A protocol entity receives messages at UDP port 161 on the system with which it is associated, except for those which report traps. Messages which report traps should be received on UDP port 162 for proper processing. An implementation of SNMP via UDP need not accept messages whose length exceeds 484 octets (bytes). However, it is recommended that implementations support larger datagrams whenever possible.

It is mandatory that all implementations of SNMP support the following five PDUs:

- GetRequest-PDU
- GetNextRequest-PDU
- GetResponse-PDU
- SetRequest-PDU
- Trap-PDU

A.3.1 How SNMP Messages Are Processed

This section describes the steps involved in the processing of SNMP messages by a protocol entity implementing SNMP. Per RFC1157 this should not be a procedure that limits *any* internal architecture.

The term *transport address* is used in this section. In the case of UDP, a transport address consists of an IP address along with a UDP port. Other transport services may be used to support SNMP. In these cases, the definition of a transport address should be made accordingly.

The top-level actions of a protocol entity which generates a message are as follows:

1. It first constructs the appropriate PDU as an ASN.1 object (for example, a GetRequest-PDU).
2. It then passes this ASN.1 object along with a community name, its source transport address, and the destination transport address, to the service which implements the desired authentication scheme. This authentication service returns another ASN.1 object.
3. The protocol entity then constructs an ASN.1 message object using the community name and the resulting ASN.1 object.
4. This new ASN.1 object is then serialized using the basic encoding rules of ASN.1, and then sent using a transport service to the peer protocol entity.

Similarly, the top-level actions of a protocol entity which receives a message are as follows:

1. It performs a rudimentary parse of the incoming datagram to build an ASN.1 object corresponding to an ASN.1 message object. If the parse fails, it discards the datagram and performs no further actions.
2. It then verifies the version number of the SNMP message. If there is a mismatch, it discards the datagram and performs no further actions.

3. The protocol entity then passes the community name and user data found in the ASN.1 message object along with the datagram′s source and destination transport addresses to the service which implements the desired authentication scheme. This entity returns another ASN.1 object or signals an authentication failure. In the latter case, the protocol entity ideally notes this failure, generates a trap, and discards the datagram (after which no further actions are performed).
4. The protocol entity then performs a rudimentary parse on the ASN.1 object returned from the authentication service to build an ASN.1 object corresponding to an ASN.1 PDU object. If the parse fails, it discards the datagram and performs no further actions. Otherwise, using the named SNMP community, the appropriate profile is selected, and the PDU is processed accordingly. If, as a result of this processing, a message is returned, then the source transport address that the response message is sent from shall be identical to the destination transport address that the original request message was sent to.

A.3.2 SNMP Protocol Data Units

In this section, we discuss the PDUs that make up the SNMP operations. Although more detail is provided here than the general explanation for SNMP operations seen in 2.10, “SNMP Operations” on page 20, fully detailed explanations of each PDU are not provided. Just enough is explained so as to understand the purpose of each PDU and what results may be expected from its use. You should refer to RFC1157 for the actual details and constructs that are involved.

Before introducing the PDU types of the SNMP protocol, it is appropriate to recall that SNMP communications are performed through the exchange of messages. We know that an SNMP message consists of a version identifier, an SNMP community name and a PDU. The SNMP operations discussed in 2.10, “SNMP Operations” on page 20, are processed in the form of a PDU.

One of ASN.1′s commonly used constructs defines an ErrorStatus field. This field is present in each PDU type and is used to include information about the result of the PDU processing. The ErrorStatus types are the following:

- noError(0)
- tooBig(1)
- noSuchName
- badValue
- readOnly
- genErr

A non-zero instance of ErrorStatus is used to indicate that an exception occurred while processing a request.

A.3.2.1 The GetRequest-PDU

The GetRequest-PDU corresponds to the GET operation and is generated by a protocol entity only at the request of its SNMP application entity.

Upon the receipt of the GetRequest-PDU, the receiving protocol entity responds according to any applicable rule in the following list:

1. If, for any object named in the GetRequest-PDU, the object′s name does not exactly match the name of some object available for GET operations in the relevant MIB view, or is an aggregate type as defined in the SMI, then the receiving entity sends to the originator of the received message a GetResponse-PDU with the ErrorStatus field set to noSuchName.
2. If the size of the GetResponse-PDU that is generated by the responding entity exceeds a local limitation, then the receiving entity sends to the originator of the received message a GetResponse-PDU with the ErrorStatus field set to tooBig.
3. If, for any object named in the GetRequest-PDU, the value of the object cannot be retrieved for reasons not covered by any of the foregoing rules, then the receiving entity sends to the originator of the received message a GetResponse-PDU with the ErrorStatus field set to genErr.
4. If none of the foregoing rules apply, meaning that the GetRequest-PDU was able to retrieve all its named objects information successfully, then the receiving entity sends to the originator of the received message a GetResponse-PDU such that, for each object named in the GetRequest-PDU, the object′s variable name and value is included. The ErrorStatus field is set to noError.

A.3.2.2 The GetNextRequest-PDU

The GetNextRequest-PDU corresponds to the GETNEXT operation and is generated by a protocol entity only at the request of its SNMP application entity.

Upon receipt of the GetNextRequest-PDU, the receiving protocol entity responds according to any applicable rule in the following list:

1. If, for any object named in the GetNextRequest-PDU, the object′s name does not lexicographically precede the name of some object available for GET operations in the relevant MIB view, then the receiving entity sends to the originator of the received message a GetResponse-PDU with the ErrorStatus field set to noSuchName.
2. If the size of the GetResponse-PDU that is generated by the responding entity exceeds a local limitation, then the receiving entity sends to the originator of the received message a GetResponse-PDU with the ErrorStatus field set to tooBig.
3. If, for any object named in the GetNextRequest-PDU, the value of the lexicographical successor to the named object cannot be retrieved for reasons not covered by any of the foregoing rules, then the receiving entity sends to the originator of the received message a GetResponse-PDU with the ErrorStatus field set to genErr.
4. If none of the foregoing rules apply, meaning that the GetNextRequest-PDU was able to retrieve all its named objects information successfully, then the receiving entity sends to the originator of the received message a GetResponse-PDU such that, for each object named in the GetNextRequest-PDU, the variable name and value of the immediate successor in the lexicographical ordering is included. The ErrorStatus field is set to noError.

A.3.2.3 The GetResponse-PDU

The GetResponse-PDU corresponds to the GET-RESPONSE operation and is generated by a protocol entity only upon receipt of the GetRequest-PDU, GetNextRequest-PDU, or SetRequest-PDU.

Upon receipt of the GetResponse-PDU, the receiving protocol entity presents its contents to its SNMP application entity.

A.3.2.4 The SetRequest-PDU

The SetRequest-PDU corresponds to the SET operation, and is generated by a protocol entity only at the request of its SNMP application entity.

Upon receipt of the SetRequest-PDU, the receiving protocol entity responds according to any applicable rule in the following list:

1. If, for any object named in the SetRequest-PDU, the object is not available for SET operations in the relevant MIB view, then the receiving entity sends to the originator of the received message a GetResponse-PDU with the ErrorStatus field set to noSuchName.
2. If, for any object named in the SetRequest-PDU, the contents of its value is not consistent with the required type of value for that variable, then the receiving entity sends to the originator of the received message a GetResponse-PDU with the ErrorStatus field set to badValue.
3. If the size of the GetResponse-PDU that is generated by the responding entity exceeds a local limitation, then the receiving entity sends to the originator of the received message a GetResponse-PDU with the ErrorStatus field set to tooBig.
4. If, for any object named in the SetRequest-PDU, the value of any object cannot be altered for reasons not covered by any of the foregoing rules, then the receiving entity sends to the originator of the received message a GetResponse-PDU with the ErrorStatus field set to genErr.
5. If none of the foregoing rules apply, meaning that the SetRequest-PDU was able to alter all of its named objects values in the relevant MIB view, then the corresponding value is assigned to the variable. Each variable assignment specified by the SetRequest-PDU should be effected as if simultaneously set with respect to all other immediate assignments specified in the same message. This means that either all variable alterations are done successfully or none of them.

   The receiving entity then sends to the originator of the received message a GetResponse-PDU with the ErrorStatus field set to noError.

A.3.2.5 The Trap-PDU

The Trap-PDU corresponds to the TRAP operation and is generated by a protocol entity only at the request of its SNMP application entity. The means by which an SNMP application entity selects the destination addresses of the SNMP application entity is implementation-specific.

Upon receipt of the Trap-PDU, the receiving protocol entity presents its contents to its SNMP application entity.

As is the case with the other PDUs, the Trap-PDU is comprised of several components. One of those components is the variable-bindings. The significance of this component is implementation-specific.

Two other components of the Trap-PDU are the generic-trap, and the specific-trap. The specific-trap provides additional information for certain cases of the generic-trap. The following are interpretations of the possible values of the generic-trap field:

coldStart (0) A coldStart trap signifies that the sending protocol entity is reinitializing itself such that the agent's configuration or the protocol entity implementation may be altered.

warmStart (1) A warmStart trap signifies that the sending protocol entity is reinitializing itself such that neither the agent configuration nor the protocol entity implementation is altered.

linkDown (2) A linkDown trap signifies that the sending protocol entity recognizes a failure in one of the communication links represented in the agent's configuration.

The Trap-PDU of type linkDown contains, as the first element of its variable-bindings, the name and value of the ifIndex instance for the affected interface.

linkUp (3) A linkUp trap signifies that the sending protocol entity recognizes that one of the communication links represented in the agent's configuration has come up.

The Trap-PDU of type linkUp contains as the first element of its variable-bindings, the name and value of the ifIndex instance for the affected interface.

authenticationFailure (4) An authenticationFailure trap signifies that the sending protocol entity is the addressee of a protocol message that is not properly authenticated. While implementations of SNMP must be capable of generating this trap, they must also be capable of suppressing the emission of such traps via an implementation-specific mechanism.

egpNeighborLoss (5) An egpNeighborLoss trap signifies that an EGP neighbor for whom the sending protocol entity was an EGP peer has been marked down and the peer relationship no longer exists.

The Trap-PDU of type egpNeighborLoss contains, as the first element of its variable-bindings, the name and value of the egpNeighAddr instance for the affected neighbor.

enterpriseSpecific (6) An enterpriseSpecific trap signifies that the sending protocol entity recognizes that some enterprise-specific event has occurred. The specific-trap field identifies the particular trap which occurred.

Appendix B. The IAB

This appendix contains information that can be a complement to understanding the administrative framework on which SNMP and the Internet community in general rely on. The IAB structure and RFCs are the main topics.

B.1 The Internet Activities Board (IAB)

As Internet research activity increased during the 1970s, it was necessary to establish an informal committee to provide technical guidance for the evolution of the protocol suite. In 1979 a group called the Internet Configuration Control Board (ICCB) was established.

In 1983, the Defense Communications Agency (DCA) declared the TCP/IP protocol suite to be standard for the ARPANET, and the ICCB was reorganized. The reorganized group was called the Internet Activities Board (IAB).

The IAB is the coordinating committee for Internet design, engineering and management. It is formed by researchers and professionals with an interest in the development of the Internet. The IAB focuses on the TCP/IP protocol suite and extensions to the Internet system to support multiple protocol suites. All IAB members are required to have at least one other major role in the Internet community in addition to their IAB membership. The IAB has a chairman which serves a term of two years. New members are appointed by the chairman of the IAB with the advice and consent of the remaining IAB members.

The IAB has the following two primary subsidiary task forces:

1. Internet Engineering Task Force (IETF)
2. Internet Research Task Force (IRTF)

Each of these task forces is led by a chairman and guided by a Steering Group.

The IETF focuses on short and mid-term protocol and architectural issues to make the Internet function properly. The IETF is a large open community of network designers, operators, vendors, and researchers, divided into eight technical areas, each with its own director. Each area has its own Working Groups to explore situations. The IETF chairman and the eight area directors make up the Internet Engineering Steering Group (IESG).

The IRTF focuses on research of TCP/IP protocols and architecture. It is formed by a community of network researchers. The IRTF is formed by a set of Research Groups (RGs), each focusing on a broad area of research. The IRTF chairman and each of the RG chairs, make up the Internet Research Steering Group (IRSG).

In the area of network protocols, the distinction between research and engineering is not always clear. Thus, it is not unusual that IETF and IRTF activities overlap. Membership overlap between the two task forces is considerable and is considered vital for cross-fertilization and technology transfer.

B.1.1 Request for Comments (RFC)

A Request for Comments (RFC) is the principal vehicle by which IAB decisions are propagated to the Internet community. The RFCs are a series of notes which were initiated in 1969 as a means for documenting the development of the original ARPANET protocol suite (RFC1000). Most RFCs are intended to promote comments and discussion, although a small proportion of RFCs document Internet standards. These in particular are marked in a *status* section to indicate one of *required*, *recommended*, *elective*, *limited use*, or *not recommended* (see Table 5 on page 248). An RFC summarizing the status of all standard RFCs is published regularly (RFC1100).

Each RFC has a number assigned to it by the RFC Editor who is a member of the IAB. Each time an existing RFC text is revised, a new RFC number is assigned. The new RFC then supersedes the older one, and this is clearly noted on the front of the newer RFC. Another member of the IAB is the Internet Assigned Numbers Authority (IANA). The IANA is responsible for managing the list of values which make up the OBJECT IDENTIFIERS used in the Internet protocol suite. For example, the IANA has assigned the number 1 to the RFC which defines the Internet standard MIB. Thus the OBJECT IDENTIFIER for this RFC is mgmt(1), or 1.3.6.1.2.1.

B.1.1.1 How to Obtain a Copy of an RFC

The RFCs can be obtained through any of the following channels:

- Printed copies are available for a modest fee from the DDN Network Information Center:

  Postal: DDN Network Information Center
  142000 Park Meadow Drive
  Suite 200
  Chantally, VA 22021
  US

  Phone: 1 800-365-3642
  1 703-802-4535

  Mail: nic@nic.ddn.mil

- In electronic form, users may use anonymous FTP (password: guest) to the host nic.ddn.mil (residing at 192.11.36.5) and retrieve files from the rfc directory.
- If your site doesn't have IP connectivity to the Internet community, but does have electronic mail access, an electronic mail message can be sent to the electronic mail address:

  `mail=server@nisc.sri.com`

  and in the subject field indicate the RFC number, for example, `Subject: SEND rfcs/rfc1130.txt`.

- If you have access to the World Wide Web, the RFCs can be obtained from the following address:

  `http://info.internet.isi.edu`

B.1.2 Functions of the IAB

As the coordinating committee for the Internet system, the IAB performs the following functions:

- Sets Internet standards
- Manages the RFC publication process
- Reviews the operation of the IETF and IRTF
- Performs strategic planning for the Internet, identifying long-range problems and opportunities
- Acts as a technical policy liaison and representative for the Internet community
- Resolves technical issues which cannot be treated within the IETF or IRTF frameworks

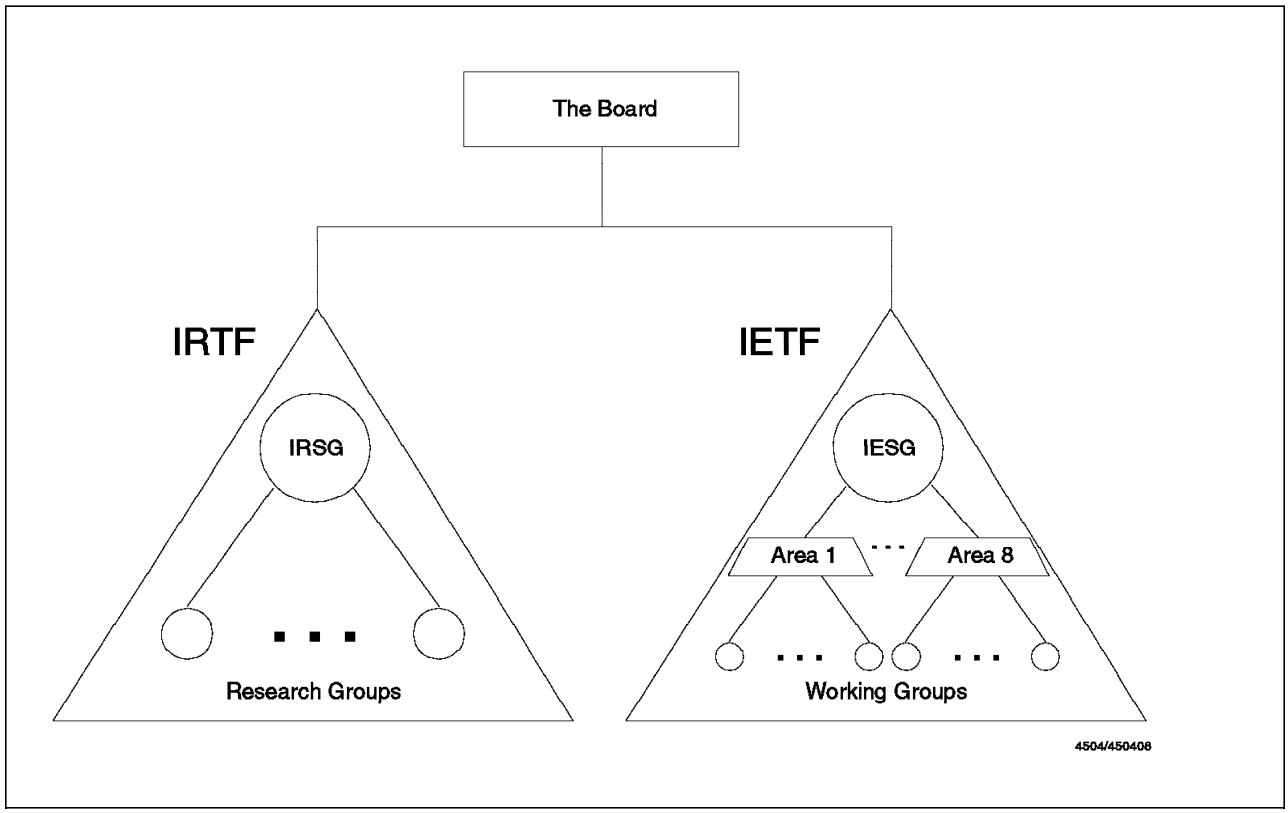


Figure 183. The IAB Organization

B.1.3 Protocol Standardization Process

The IAB provides standards with the intention of coordinating the evolution of the Internet protocols. With the increasing use of the Internet protocols for commercial purposes, standards coordination has become even more important.

Protocols which are to become standards in the Internet go through a series of states involving increasing amounts of scrutiny and experimental testing. At each step, the IESG of the IETF must make a recommendation for advancement of the protocol and the IAB must ratify it. This process is referred to as the *Standards Track*. If a recommendation is not ratified, the protocol is submitted

again to the IETF for further work. Table 4 on page 248 lists the Internet protocol state definitions.

Table 4. Internet Protocol State Definitions

Protocol State	Definition
Standard Protocol	The IAB has established this as an official standard protocol for the Internet.
Draft Standard Protocol	The IAB is actively considering this protocol as a possible Standard Protocol.
Proposed Standard Protocol	These are protocol proposals that may be considered by the IAB for standardization in the future.
Experimental Protocol	Typically, experimental protocols are those that are developed as part of an ongoing research project not related to an operational service offering. An experimental protocol may sometimes mean that the protocol is not intended for operational use. A system should not implement an experimental protocol unless it is participating in the experiment and has coordinated its use of the protocol with the developer of the protocol.
Historic Protocol	These are protocols that are unlikely to ever become standards in the Internet either because they have been superseded by later developments or due to lack of interest.

Table 5 lists the Internet protocol status definitions mentioned in B.1.1, “Request for Comments (RFC)” on page 246.

Table 5 (Page 1 of 2). Internet Protocol Status Definitions

Protocol Status	Definition
Required Protocol	A system must implement the required protocols.
Recommended Protocol	A system should implement the recommended protocols.
Elective Protocol	A system may or may not implement an elective protocol. The general notion is that if you are going to do something like this, you must do exactly this. There may be several elective protocols in a general area. For example, there are several electronic mail protocols, and several routing protocols.
Limited Use Protocol	These protocols are for use in limited circumstances. This may be because of their experimental state, specialized nature, limited functionality, or historic state.

Table 5 (Page 2 of 2). Internet Protocol Status Definitions	
Protocol Status	**Definition**
Not Recommended Protocol	These protocols are not recommended for general use. This may be because of their limited functionality, specialized nature, or experimental or historic state.

Appendix C. Problem Determination

This appendix is intended to be used for determining solutions to problems encountered while using SNMP. It is divided into the following areas:

- Problem Determination for Starting OS/400 SNMP
- Problem Determination for SNMP Agent
- Problem Determination for SNMP Manager APIs
- Problem Determination for SNMP Trap Manager
- Problem Determination for SNMP Subagent APIs

C.1 Problem Determination for Starting OS/400 SNMP

To start SNMP you have to issue one of the following commands:

STRTCP (If SNMP is configured for automatic start)

STRTCPSVR SERVER(*SNMP) (If SNMP is not configured for automatic start)

To verify that SNMP has started correctly you can use the command WRKACTJOB. OS/400 SNMP starts the following jobs:

- QTMSNMP - SNMP agent processing job
- QTMSNPRCV - receives incoming SNMP messages on port 161 and passes them to QTMSNMP for processing
- QSNMPSA - SNMP subagent job

These jobs can be found running under the QSYSWRK subsystem using the WRKACTJOB(QSYSWRK) command. See Figure 184 on page 252.

```
                      Work with Active Jobs                    RALYAS4A
                                                          07/27/95  16:14:15
 CPU %:  8.8      Elapsed time:  00:01:36       Active jobs:  70

 Type options, press Enter.
  2=Change  3=Hold  4=End  5=Work with  6=Release  7=Display message
  8=Work with spooled files  13=Disconnect ...

Opt Subsystem/Job User        Type  CPU %  Function     Status
    QSYSWRK       QSYS        SBS      .0                DEQW
     QAPPCTCP     QSYS        BCH      .0  PGM-QZPAIJOB  TIMW
     QECS         QSVSM       BCH      .0  PGM-QNSECSJB  DEQW
     QMSF         QMSF        BCH      .0                DEQW
     QNSCRMON     QSVSM       BCH      .0  PGM-QNSCRMON  DEQW
     QSNMPSA      QTCP        BCH      .0  PGM-QNMSARTR  DEQW
     QTCPIP       QTCP        BCH      .0                DEQW
     QTMSNMP      QTCP        BCH      .0  PGM-QTOSMAIN  DEQW
     QTMSNPRCV    QTCP        BCH      .0  PGM-QTOSRCVR  DEQW

 Parameters or command                                    More...
 ===>
 F3=Exit    F5=Refresh  F10=Restart statistics  F11=Display elapsed data
 F12=Cancel   F23=More options   F24=More keys
```

Figure 184. SNMP Agent Jobs Running Under QSYSWRK

If you issued the STRTCP command to start SNMP and you cannot see the jobs QSNMPSA, QTMSNMP and QTMSNMPRCV running under subsystem QSYSWRK, check the following:

- Verify that the AUTOSTART parameter is set to *YES in the SNMP attributes.
- Make sure that no other application is trying to use UPD port 161. To verify this, issue the following command:

  NETSTAT

  Select option 3 (Work with TCP/IP Connections)

  Figure 185 on page 253 shows that SNMP is currently active.

```
                     Work with TCP/IP Connection Status
                                                        System: RALYAS4A
Local internet address  . . . . . . . . . . . :   *ALL

Type options, press Enter.
  4=End   5=Display details

     Remote           Remote     Local
Opt  Address          Port       Port       Idle Time  State
     *                *          ftp-con >  003:44:28  Listen
     *                *          telnet     003:44:26  Listen
     *                *          snmp       000:03:22  *UDP
     *                *          APPCove >  003:44:42  Listen
     *                *          APPCove >  003:44:41  *UDP
     *                *          lpd        003:44:01  Listen
     *                *          1025       000:03:22  *UDP




F5=Refresh   F11=Display byte counts   F13=Sort by column
F14=Display port numbers   F22=Display entire field   F24=More keys
```

Figure 185. Working with TCP/IP Connections Status

To see the port numbers associated with each port name you should press F14. Figure 186 shows the port number associated with SNMP port name.

```
                     Work with TCP/IP Connection Status
                                                        System: RALYAS4A
Local internet address  . . . . . . . . . . . :   *ALL

Type options, press Enter.
  4=End   5=Display details

     Remote           Remote  Local
Opt  Address           Port    Port  Idle Time  State
     *                    *     21  003:44:28  Listen
     *                    *     23  003:44:26  Listen
     *                    *    161  000:03:22  *UDP
     *                    *    397  003:44:42  Listen
     *                    *    397  003:44:41  *UDP
     *                    *    515  003:44:01  Listen
     *                    *   1025  000:03:22  *UDP




F5=Refresh   F11=Display byte counts   F13=Sort by column
F14=Display port names   F15=Subset by local address   F24=More keys
```

Figure 186. Port Numbers Associated with Port Names

Port 161 is used by SNMP to receive requests; no other application should use port 161. Port 161 is the well-known port for SNMP. If another application is using port 161, end it and start SNMP again.

C.2 Problem Determination for SNMP Agent

Problem:

- If SNMP managers are not receiving any responses from the OS/400 SNMP agent for any SET, GET or GETNEXT, the following should be checked:

  Solution:

  – Make sure SNMP is active on the AS/400, see C.1, “Problem Determination for Starting OS/400 SNMP” on page 251.

  – Make sure the link between the SNMP manager and the SNMP agent is active. If the link is TCP/IP or AnyNet you can issue the command:

    NETSTAT

    Select option 1 (Work with TCP/IP Interface Status)

```
                     Work with TCP/IP Interface Status
                                                        System: RALYAS4A
Type options, press Enter.
  5=Display details   8=Display associated routes   9=Start   10=End
  12=Work with configuration status

     Internet         Network             Line      Interface
Opt  Address          Address          Description  Status
     9.24.104.56      9.24.104.0       L41TR        Active
     9.67.60.20       9.67.60.0        *IPS         Active
     127.0.0.1        127.0.0.0        *LOOPBACK    Active









F3=Exit   F4=Prompt   F5=Refresh   F11=Display line information   F12=C
F13=Sort by column    F24=More keys
```

Figure 187. Status of Native TCP/IP Interfaces and AnyNet

    You can identify the link by the internet address. If the link is a native TCP/IP one, the line description name will be displayed in the Line Description field. If the link is not native TCP/IP (Sockets over SNA), *IPS will be displayed in the Line Description field. The Interface Status field shows the status of the link. If the link is not available (Inactive), option 9 (Start) can be used to make it available.

    Another way to verify that the link is active is to use the PING command. Refer to 4.2, “Verifying a TCP/IP Connection” on page 31 on how to use this command.

  – Make sure that the internet address is that of the OS/400 SNMP agent you intend to send a request to.

  – Make sure the SNMP manager does not compare the response PDU source IP address with the destination IP address in the PDU it sent. The

SNMP agent will send the response PDU to the IP address which sent the original PDU. If the SNMP agent has more than one link via which to send a response, it will choose one (maybe not the same IP address that the request was sent to).

- Make sure OS/400 SNMP is using port 161 to receive requests. The OS/400 SNMP agent uses UDP port 161 to receive requests from managers. It doesn't use UDP port 161 to send response PDUs to those requests, due to technical reasons. The SNMP management application should not check the response PDU port number. The SNMP manager should not compare the source UDP port number in the response PDU with the destination UDP port number in the request PDU it sent to the OS/400 SNMP agent.
- Make sure that the SNMP manager is specifying a community name that is known to the OS/400 SNMP agent. The community names available in the OS/400 SNMP agent can be displayed using the following command:

  CFGTCPSMNP

  Select option 2 (Work with Communities for SNMP) and you'll be able to display a list of SNMP communities configured.

  Considerations about communities:

  Community names are case sensitive, so make sure that the SNMP manager is specifying the community name correctly (for example: PUBLIC and public are two different communities names).

  If the SNMP manager is an ASCII system, the Translate Community name (ASCIICOM) parameter for the OS/400 community should be *YES. If the SNMP manager is an EBCDIC system or the community name has one or more characters that can not be displayed, the ASCIICOM parameter for the OS/400 community should be *NO. If the community name or the ASCIICOM parameter needs to be changed, the community must be removed by using the RMVCOMSNMP command, then add a new community with the correct values using the ADDCOMSNMP command. Make sure that the value for the Translate Community name (ASCIICOM) parameter for the OS/400 community corresponds to the community name being specified by the SNMP manager.
- Make sure that the IP address of the SNMP manager is defined in the OS/400 community. The SNMP manager may be specifying a community name that is known to the OS/400 SNMP agent, but the IP address of the SNMP manager may not be part of the OS/400 community it specified. If the system that the manager is running on has multiple IP addresses, ensure that the IP address used to send UDP packets to the AS/400 agent is the one included in the IP addresses for the community name configuration intended for the manager. The IP address of the SNMP manager can be added to the community by using the CHGCOMSNMP command.
- Make sure that the object access for the community is either *WRITE or *READ. The object access can be changed using the CHGCOMSNMP command. If the parameter OBJACC (object access) has a *WRITE value then SET operations can be issued; if it has a *READ value, then only the GET and GETNEXT operations can be issued; if it has a *NONE value,

then the manager is not able to access the community, if it has an *SNMPATR value, the value of the OBJACC parameter of the SNMP attributes will be the one used (the SNMP attributes are shipped with the OBJACC parameter set to *NONE).

Problem:

- If SNMP managers are receiving error messages from the OS/400 SNMP agent for any GET or GETNEXT requests, the following should be checked:

  Solution:

  – Make sure the SNMP manager is specifying an object identifier for an object that is supported by the OS/400 SNMP agent. Verify that the SNMP manager is specifying a correct object identifier in the request and that the object identifier being specified is supported by the OS/400 SNMP agent.

  Problem:

- If SNMP managers are receiving error messages from the OS/400 SNMP agent for SET requests, the following should be checked:

  Solution:

  – Make sure that the SNMP manager is specifying an object identifier for an object that is supported by the OS/400 SNMP agent. Verify that the SNMP manager is specifying a correct object identifier in the request and that the object identifier being specified is supported by the OS/400 SNMP agent.

  – Make sure that the SNMP manager is specifying an object identifier for an object that can be changed. Verify that the object identifier that the SNMP manager is attempting to set is not defined as read-only.

  – Make sure the SNMP manager is specifying a valid value for the object.

  – Make sure that the object access for the community is *WRITE. The object access can be changed using the CHGCOMSNMP command. If the parameter OBJACC (object access) has a *WRITE value, then SET operations can be issued; if it has a *READ value, then only GET and GETNEXT operations can be issued; if it has a *NONE value, then the manager is not able to access the community; if it has an *SNMPATR value, the value of the OBJACC parameter of the SNMP attributes will be the one used (the SNMP attributes are shipped with the OBJACC parameter set to *NONE).

  Problem:

- If SNMP managers are unable to access any objects found in the APPN MIB, Client Management MIB, or NetView/6000 MIB, the following should be checked:

  Solution:

  – Make sure that the OS/400 subagent job is running. The OS/400 subagent runs in job QSMNPSA in subsystem QSYSWRK. End the SNMP agent by using the ENDTCPSRV (*SNMP) command; this will end all the jobs associated with the SNMP agent in subsystem QSYSWRK and restart the SNMP agent by using the STRTCPSRV command. Use the command WRKACTJOB to determine if this job is active.

Problem:

- If traps are not being received by an SNMP manager, the following should be checked:

  Solution:

  - Make sure that the IP address of the SNMP manager to receive the trap is specified correctly in the OS/400 SNMP attributes. The IP address of the SNMP manager can be changed using the CHGSNMPA command. The correct address should be specified in the TRPMGR parameter.
  - Make sure that the trap community name specified is recognized by the SNMP manager. The trap community name of the SNMP manager can be changed using the CHGSNMPA command. The correct trap community name should be specified in TRPMGR parameter.

    The SNMP agent has the capability to log traps (in journal QSNMP in library QUSRSYS). The log trap parameter can by changed using the CHGSNMPA command. Verify (using the OS/400 SNMP logging capabilities) that the expected trap PDUs are being sent by the OS/400 SNMP agent to the correct SNMP manager. See chapter Chapter 12, “Journal for SNMP Logging” on page 211 for information on how to use this function.

Problem:

- If authenticationFailure traps are not being received by an SNMP manager, the following should be checked:

  Solution:

  - Make sure that the IP address of the SNMP manager to receive the trap is specified correctly in the OS/400 SNMP attributes. The IP address of the SNMP manager can be changed using the CHGSNMPA command. The correct address should be specified in the TRPMGR parameter.
  - Make sure that the trap community name specified is recognized by the SNMP manager. The trap community name of the SNMP manager can be changed using the CHGSNMPA command. The correct trap community name should be specified in the TRPMGR parameter.
  - Make sure that OS/400 SNMP is enabled to send authenicationFailure traps. To enable OS/400 SNMP to send them, the CHGSNMPA command should be used. *YES should be specified in the SNDAUTTRP parameter of the CHGSNMPA command.

    Verify (using the OS/400 SNMP logging capabilities) that the expected trap PDUs are being sent by the OS/400 SNMP agent to the correct SNMP manager. See Chapter 12, “Journal for SNMP Logging” on page 211 for information on how to use this function.

Problem:

- If entries are not being made in journal QSNMP in library QUSRSYS for SET requests received from the SNMP manager, the following should be checked:

  Solution:

  - Make sure that OS/400 SNMP is enabled to log SET requests. This capability is enabled by using the CHGCOMSNMP command and specifying *YES in the LOGSET parameter or by using the CHGCOMSNMP command and specifying *SNMPATR in the LOGSET parameter and specifying *YES in the LOGSET parameter for SET requests in OS/400 SNMP attributes by using CHGSNMPA command.

Problem:

- If entries are not being made in journal QSNMP in library QUSRSYS for GET or GETNEXT requests received from the SNMP manager, the following should be checked:

  Solution:

  – Make sure that OS/400 SNMP is enabled to log GET requests. This capability is enabled by using the CHGCOMSNMP command and specifying *YES in the LOGGET parameter or by using the CHGCOMSNMP command and specifying *SNMPATR in the LOGGET parameter and specifying *YES in the LOGGET parameter in OS/400 SNMP attributes by using CHGSNMPA command.

  Problem:

- If entries are not being made in journal QSNMP in library QUSRSYS for traps sent to the SNMP manager, the following should be checked:

  Solution:

  – Make sure that OS/400 SNMP is enabled to log traps. This capability is enabled by using the CHGSNMPA command and specifying *YES in the LOGTRP parameter of the OS/400 SNMP attributes.

  Problem:

- If an SNMP agent job is getting numerous TCP4017 messages, then the following should be checked:

  Solution:

  – Make sure that the remote name server values are correct.

  – Make sure that the complete local domain name is specified for AS/400 (option 12 of CFGTCP).

  Problem:

- If SNMP agent job is not starting (with the first error in the job log CPF7003) then following action should be taken:

  Solution:

  – Turn off the logging for each SNMP community. Stop then restart the SNMP agent using the ENDTCPSVR and STRTCPSVR commands. Delete the journal receivers (DLTJRN JRN(QUSRSYS/QSNMP)) and the journal (DLTJRNRCV JRNRCV(QUSRSYS/QSNMP*)). To complete the procedure, use the CFGTCPSNMP command to restart SNMP logging.

  – Make sure, that the complete local domain name is specified for the AS/400 (option 12 of CFGTCP).

C.3 Problem Determination for SNMP Manager APIs

- If the SNMP management application received an out of memory or out of buffers error, it is because the SNMP API has run out of resources to complete the operation. To solve this, the management application can try reducing the size of requested operations. If the problem persists, the error should be reported using the ANZPRB command.
- If the SNMP management application received an out of varBinds error, it means the requested operation has exceeded the allowable number of varBinds for a single operation. In order to correct this, management application can try reducing the number of varBinds in the varBind list.
- If the SNMP management application received an invalid OID (object identifier) error, it is due to not specifying the correct dotted decimal notation for an object identifier in the varBind list. In order to solve this, the management application should specify the object identifier in the correct decimal notation.
- If the SNMP management application received an invalid value error, it is because the SNMP APIs will do some rudimentary checking on the value supplied during snmpSet operation, the APIs will detect an incorrect length in integer types, and will also check for incorrect dotted decimal notation on IP addresses and OIDs when supplied as values. In order to avoid this, the management application should specify the value on a set in the correct form as specified by the ASN type.
- If the SNMP management application received an invalid value representation error, it is because the ASN type specified for a particular OID in the varBind list is not recognized as common ASN type. In order to avoid this, the management application should specify a known ASN type as listed in the QTOMEAPI CLEINC file in QSYSINC library.
- If the SNMP management application received an encode or decode error, it is because the PDU specified could not be encoded or decoded for transmission across the wire. In order to solve this, the command should be retried, and if the problem persists the error should be reported using the ANZPRB command.
- If the SNMP management application received an invalid community name error, it is due to the value in the community name length field not being in the range of 1 to 256. To avoid this, the management application should specify a community name length that is greater than 0 and less than or equal to 256.
- If the SNMP management application received a timeout parameter error, it is due to the value in the timeout parameter field not being in the range of 1 to 100. To avoid this, the management application should specify a timeout value that is greater than 0 and less than or equal to 100.
- If the SNMP management application received an unknown host error, it is due to a host name being specified that is not recognized as a valid host on the network. In order to solve this, the management application should specify the correct host name or specify the correct dotted decimal notation of the IP address.
- If the SNMP management application received a not OK error, it is because the value length field on a varBind in the varBind list was less than zero. In order to solve this, the management application should specify a value length field greater than or equal to zero.

- If the SNMP management application received a timeout from the SNMP APIs the causes may be several such as:
  - The timeout value being set too low. In order to solve this, the timeout value should be increased and then try the management application again.
  - The SNMP agent receiving the SNMP operation is not active. In order to solve this, make sure the receiving agent is active.
  - The SNMP management application is specifying a correct community name but the IP address of the manager is not part of the community. In order to avoid this, make sure the IP address is known to the SNMP agent as a valid SNMP manager.
  - A problem with the TCP/IP or AnyNet network. In order to solve this, the TCP/IP Configuration and Reference manual should be referred for information about network problem determination.
- If the SNMP management application received a sockets error while trying to initiate an SNMP operation, this is because the SNMP API did not connect properly to sockets to perform its operation across the TCP/IP or AnyNet network. In order to avoid this, make sure that the TCP/IP and AnyNet support is active in the system and then retry the command.
- If the SNMP management application received an invalid PDU type, this is due to the management application trying to issue an SNMP operation of one type while specifying a PDU built for another type. In order to to solve this, the PDU type must match that of the SNMP operation being performed.
- If the SNMP management application received an invalid IP address, this is because the management application tried to issue an SNMP operation with an invalid IP address. In order to solve this, the IP address specified should be the correct one and be specified in either the host name form or in the dotted decimal form.
- If the SNMP management application received a Domain Error, Invalid pointer, or Invalid pointer type return code from the SNMP APIs, this is due to the management application specifying a pointer which caused an object domain error, referencing a location in a space that does not contain a pointer, or the pointer referenced storage in system state. These errors are equivalent to escape messages MCH6801, MCH3601, MCH3602. In order to avoid this, the management application should not use system state reference; retry the operation passing in a valid pointer to user state storage.
- If the SNMP management application received an invalid PDU, host, community or invalid pointer type return code from the SNMP APIs, this is because the application specified a NULL pointer. In order to avoid this, the management application should use a non-NULL pointer.
- If the SNMP management application received a return code 1 from the APIs, this states that the amount of space allocated for the value returned in one or more varBinds, for a GET or GETNEXT operation was not sufficient. In order to solve this, the management application should specify a greater value in the val_len field of the _varBind structure.

- If the SNMP management application received a non-zero returned status in the Error Status field of SNMP PDU on a GET or GETNEXT operation, this is caused by the following:
  - The SNMP agent could not fit the contents of the returned data in the SNMP operation. In order to solve this, if the management application specified more than one object identifier to retrieve, then the number of the object identifier should be reduced, until the returned data fits into an SNMP message.
  - The SNMP management application is specifying an object identifier incorrectly in the request or it is specifying an object identifier that is not supported by the receiving SNMP agent or its subagent. In order to avoid this, the SNMP management application should specify an object identifier for an object that is supported by the receiving SNMP agent or its subagents.
- If the SNMP management application received a non-zero return status in the Error Status field of SNMP PDU on a SET operation, this is caused by the following:
  - The SNMP management application is specifying an incorrect object identifier in the request or it is specifying an object identifier that is not supported by the receiving SNMP agent or its subagents. In order to avoid this, the SNMP management application should specify an object identifier for an object that is supported by the receiving SNMP agent or its subagents.
  - The SNMP management application is attempting to set an object that is defined as read-only. In order to avoid this, the SNMP management application should specify an object identifier for an object that can be changed.
  - The SNMP management application is attempting to set an object value that is not valid. The value could have the incorrect ASN syntax or it may not be in the acceptable range of values. In order to solve this, the SNMP management application should specify the correct ASN syntax in the ASN type field and specify a value within the acceptable range.

C.4 Problem Determination for SNMP Trap Manager

Problem:

- If the SNMP trap manager jobs do not start when the STRTRPMGR command is issued, the following should be checked:

  Solution:

  – Make sure no other applications are using port 162. The trap manager uses port 162 to receive all traps and then forward them. Use the command NETSTAT and select option 3 (Work with TCP/IP Connections) to verify no other application is using port 162. If another application is using port 162, end it and start the SNMP trap manager using the STRTRPMGR command.

  Problem:

- If traps are not being received by the SNMP trap manager, the following should be checked:

  Solution:

  – Make sure the IP address specified in the SNMP agent system sending the trap is the one corresponding to this system.

  Problem:

- If traps are not being forwarded to other SNMP manager systems, the following should be checked:

  Solution:

  – Make sure that the IP address of the SNMP manager to receive the trap is specified correctly in the OS/400 SNMP attributes. The IP address of the SNMP manager can be changed using the CHGSNMPA command. The correct address should be specified in the TRPMGR parameter.

  – Make sure that the trap community name specified is recognized by the SNMP manager. The trap community name of the SNMP manager can be changed using the CHGSMPA command. The correct trap community name should be specified in the TRPMGR parameter.

  – Make sure the trap forwarding capability was enabled when the trap manager was started. To enable the trap forwarding capability *YES should be specified in the FWDTRP parameter in the STRTRPMGR command.

  – Make sure the SNMP agent is active. The OS/400 SNMP agent is active if the jobs QTMSNMP and QTMSNMPRCV are running under the subsystem QSYSWRK. The WRKACTJOB command can be used to determine if these jobs are active.

  Problem:

- If jobs QTMSNMP, QTMSNMPRCV, QTRAPMGR and QTRAPMGRRCV in subsystem QSYSWRK are consuming a large amount of processor resource, the following should be checked:

  Solution:

  – Make sure there are no rings of trap managers in the network. It is possible that this system is part of a ring of trap managers in your network that forward traps to each other, generating a trap loop.

- Make sure that the IP address of the trap manager being specified is not the one of this system. If trap forwarded was specified and the IP address of the trap manager system specified in the OS/400 SNMP attributes is the IP address of this system, then you should end the trap manager by using the ENDTRPMGR command and restart it using the STRTRPMGR command specifying *NO in the FWDTRP parameter. Or you can remove the IP address of this system from the SNMP attributes by using the command CHGSNMPA.

C.5 Problem Determination for SNMP Subagent APIs

Problem:

- If the SNMP subagent cannot connect to the SNMP agent, the following could be the causes:

  Solution:

  - The SNMP agent job is not running. In order to solve this, the SNMP agent job (QTMSNMP) should be running under subsystem QSYSWRK. This can be verified using the WRKACTJOB command; another way to verify this is to issue a send SNMP GET request for some easy OID using an SNMP manager or a utility such as snmpinfo.
  - A return code of snmpsa_RC_timeout was returned by connectSNMP(). In order to solve this, the duration of the timeout (third parameter) should not be too small, with respect to system load. Generally, 20 or so should be sufficient.
  - A return code of snmpsa_RC_parmerr was returned by connectSNMP(). In order to solve this, the data queue and library name parameters should not be NULL pointers and they should be between 1 and 10 bytes in length and valid library and queue names should be specified. The library name should not be QTEMP.

Problem:

- If the SNMP subagent is not able to successfully perform the open function, the following could be the causes:

  Solution:

  - The subagent receives a NULL pointer from the mkDPIopen() routine. A NULL pointer from the mkDPIopen() is usually caused by a simple error in one of the parameters. In order to avoid this, each of the parameters in the call should be specified correctly and the subagent OID should not violate SNMPv2 limits; no more than 128 subIDs and each subID must be representable in a 32-bit unsigned integer field.
  - The subagent cannot send the DPI open packet. If this condition arises, it is important to verify that a call has successfully been made to the connectSNMP() routine prior to calling sendDPIpacket().
  - The subagent received an SNMP_ERROR_DPI_othererror (101) in the DPI response packet from the agent. This occurs due to some invalid value in the DPI open packet. For example, the subagent′s OID must be NULL terminated and end with a subID, not a ″.″.
  - The subagent received an SNMP_ERROR_DPI_duplicate-Subagentidentifier (109) in the DPI response packet from the agent. This occurs when the SNMP agent already has an active subagent with same OID, and the subagent MIB saAllowDuplicateIDs is set to 2. In order to solve this, the OID the subagent uses in the mkDPIopen() should be changed or the subagent MIB saAllowDuplicateIDs should be changed to 1 to allow duplicates.

Problem:

- If the SNMP subagent is not able to successfully perform the register function, the following could be the causes:

  Solution:

  - The subagent receives a NULL pointer from the mkDPIregister() routine. A NULL return from the mkDPIregister() indicates some parameter error. In order to solve this, the subagent's log should be checked for exceptions. If any are found, they should be corrected; the subagent code should be rebuilt and the call should be tried again. If no exceptions are found, the parameters should be verified.
  - The subagent received an SNMP_ERROR_DPI_otherError (101) in the DPI response packet from the agent. The most common cause of this error in response to a DPI register packet is that the subtree OID did not end with ".". In order to avoid this, a subtree OID should end with a ".".
  - The subagent received an SNMP_ERROR_DPI_alreadyRegistered (103) in the DPI response packet from the agent. The most common cause of this error is that the subagent attempted to register a subtree that would cause a protected subtree to be affected. In order to solve this, the subtree to be registered should not be above, at, or within one of the SNMP agent's protected subtrees.
  - The subagent received an SNMP_ERROR_DPI_highPriorityRegistered (104) in the DPI response packet from the agent. In order to solve this, the subtree registration should be re-requested with a higher priority or zero (which requests the highest). If the already registered subagent has priority zero (get the subagent MIB saTstatus for the subtree), then there is no higher priority to request, so the other subtree should be unregistered or its subagent ended before the subtree can be registered.

Problem:

- If the SNMP subagent has successfully done contact, open and register, but does not receive any DPI packets (from the SNMP agent) the following can be the possible causes:

  Solution:

  - The SNMP agent job is not running. In order to solve this, the SNMP agent job (QTMSNMP) should be running under subsystem QSYSWRK and responding to PDUs in a normal way. This can be verified using the WRKACTJOB command.
  - There have not been any SNMP PDUs for the subtrees registered by the subagent. In order to avoid this, it should be verified that an SNMP PDU (GET, GETNEXT or SET) for an OID in the subagent's subtree has been sent to the SNMP agent.
  - The subagent or the subagent's subtree registration status is not valid (1). In order to solve this, the status OIDs in the subagent tree should have a valid value (1). These OIDs are in the SNMP subagent MIB in the saTable and saTreeTable, and are the saStatus OID (for the right subagent) and saTstatus OID (for the right subtree). (The ASN.1 definitions for the subagent MIB can be found in QSYS/QANMMIB, member IBMSNMPSA.)

Problem:

- If the SNMP subagent works for a while, but then mkDPIset() returns NULL when trying to build structure for a response packet, this means there is insufficient memory to build the internal structure for the DPI packet because it has not been freed after having been dynamically allocated. In order to solve this, the fDPIset() routine should be called so that memory use does not monotonically increase while the subagent is running. If some other exception occurred, messages in the joblogs for the SNMP agent job (QTMSNMP) and subagent job should be looked for, corrected and the operation retried again.

  Problem:

- If the SNMP subagent works for a while, but then pDPIpacket() returns NULL when trying to parse a new incoming package, this means there is insufficient memory to build the internal structure for the DPI packet, because it has not been freed, after having been dynamically allocated. In order to solve this, the fDPIparse() routine should be called so that memory use does not monotonically increase while the subagent is running. If some other exception occurred, messages in the joblogs for the SNMP agent job (QTMSNMP) and subagent job should be looked for, corrected and the operation retried again.

  Problem:

- If the SNMP subagent occasionally gets DPI unregister or close packet from the agent with reason_code of SNMP_UNREGISTER_timeout or SNMP_CLOSE_timeout, this means the subagent has taken too long to respond to some SNMP request. That is, longer than the timeout value used by the subagent in the mkDPIopen() or mkDPIregister() calls. In order to solve this, re-open or re-register with a longer timeout or a smaller max_varBinds value used in the mkDPIopen() call.
  Important note: the entire SNMP agent waits for up to the timeout value for a response to each DPI request. If requests for a particular OID or subagent subtree takes a long time (relatively) for the subagent to process, then consideration should be given to registering that OID or subtree separately (by the same subagent), with a appropriately longer timeout.

  Problem:

- The SNMP subagent gets snmpsa_RC_err return code which is caused by a run-time exception, which is not covered by some other, more specific, return code. In order to solve this, messages in the joblogs for both the SNMP agent job (QTMSNMP) and the subagent job (that received the snmpsa_RC_err return code) should be checked for exceptions, corrected and the subagent API calls retried. (By its nature this return code is fairly rare, and the cause for the exception is usually obvious in either the agent or the subagent job.)

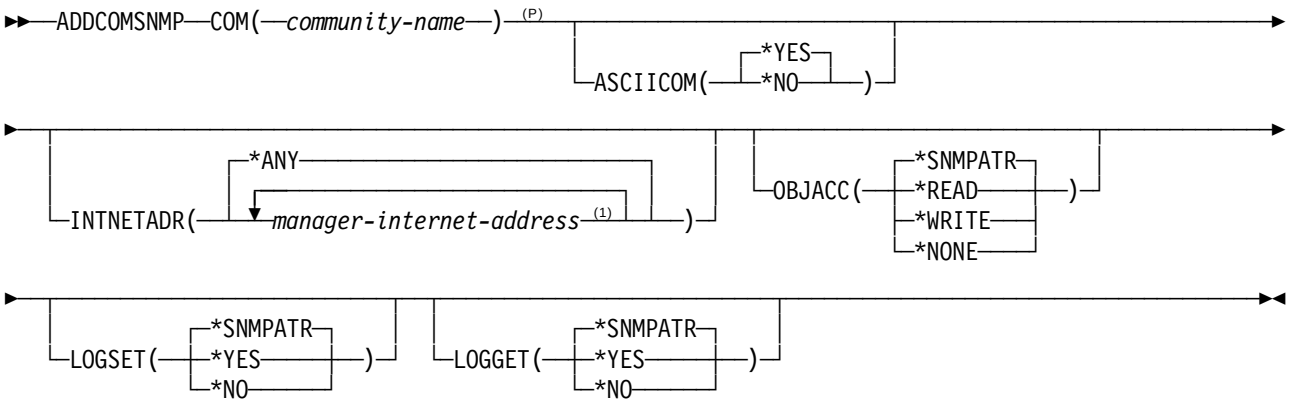
Appendix D. CL Commands for AS/400 SNMP

This appendix provides syntax diagrams and explanations of the CL commands for AS/400 SNMP.

D.1 ADDCOMSNMP (Add Community for SNMP) Command

The format of the ADDCOMSNMP command is as follows:

Job: B,I Pgm: B,I REXX: B,I Exec



Notes:

P All parameters preceding this point can be specified in positional form.

1 A maximum of 300 repetitions.

D.1.1 Purpose

The Add Community for SNMP (ADDCOMSNMP) command defines an SNMP community profile and adds it to the SNMP agent community list. An SNMP agent uses a community profile to determine whether or not to honor a request sent by an SNMP manager. The community profile consists of a community name, an object access specification, and a list of the SNMP managers that are part of the community. The combination of the community name (COM) and the translate to ASCII community (ASCIICOM) parameters defines a community.

Multiple community profiles, each having a unique community name, may exist in the SNMP agent community list at one time. Similarly, the same internet address may appear in more than one community profile.

The OS/400* SNMP agent does not support community views. A view is a subset of the objects in the Management Information Base (MIB). Each OS/400 community consists of all of the objects in the MIB.

Restrictions: An SNMP manager sends three types of requests: GET, GETNEXT, and SET. GET and GETNEXT requests are used to read management information base (MIB) variables and a SET request is used to modify MIB variables. For a request from an SNMP manager to be accepted by the AS/400 SNMP agent, all of the following must be true:

1. The community name in the SNMP manager request specifies a defined community.
2. The internet address of the manager that sent the request must be listed in the community profile.
3. For a SET request, the community object access must allow write operations to occur. For a GET request or GETNEXT request, read operations must be allowed.
4. For a SET request, the object specified in the request must be able to be changed. For a GET request or GETNEXT request, the object must be readable.

D.1.2 Required Parameter

COM

This specifies the name of the SNMP community being added. Each SNMP community name must be unique.

Community-name: Specify the name of the SNMP community being added. The name may contain characters that cannot be displayed (for example, X′60619E′).

D.1.3 Optional Parameters

ASCIICOM

Specifies whether the community name is translated to ASCII characters when the community profile is added to the SNMP agent community list.

***YES:** The community name is translated to ASCII characters when the community profile is added to the SNMP agent community list. This value should be specified if the SNMP manager system defines its community names entirely of ASCII characters. An error message is sent if the community name cannot be translated to ASCII characters.

***NO:** The community name is not translated to ASCII characters when the community profile is added to the SNMP agent community list. This value should be specified if the SNMP manager system defines its community names using EBCDIC characters or characters that cannot be displayed.

INTNETADR

Specifies the internet addresses of the SNMP managers that are part of this community.

***ANY:** Allow any SNMP manager to be part of this community.

Manager-internet-address: Specify the internet address of the SNMP manager. The internet address is specified in the form *nnn.nnn.nnn.nnn*, where *nnn* is a decimal number ranging from 0 through 255. An internet address is not valid if it has a value of all binary ones or all binary zeros for the network identifier (ID) portion or the host ID portion of the address. If the internet address is entered from a command line, the address must be enclosed in apostrophes. Up to 300 unique internet addresses may be specified. The same internet address may appear in more than one community profile.

OBJACC

Specifies the object access for the community.

***SNMPATR:** The object access defined with the Change SNMP Attributes (CHGSNMPA) command is used for this community.

***READ:** Allow SNMP managers that are part of this community to read all management information base (MIB) objects with GET or GETNEXT requests. Modification of MIB objects by SNMP managers is not permitted.

***WRITE:** Allow SNMP managers that are part of this community to change all MIB objects that are able to change with SET requests. Specifying *WRITE implies *READ access.

***NONE:** Do not allow SNMP managers that are part of this community any access to MIB objects.

LOGSET

This specifies whether SET requests from SNMP managers in this community are logged in journal QSNMP in library QUSRSYS.

***SNMPATR:** The value defined with the Change SNMP Attributes (CHGSNMPA) command is used for this community.

***YES:** SET requests are logged.

***NO:** SET requests are not logged.

LOGGET

This specifies whether GET requests and GETNEXT requests from SNMP managers in this community are logged in journal QSNMP in library QUSRSYS.

***SNMPATR:** The value defined with the Change SNMP Attributes (CHGSNMPA) command is used for this community.

***YES:** GET requests and GETNEXT requests are logged.

***NO:** GET requests and GETNEXT requests are not logged.

This is an example of the use of the ADDCOMSNMP command:

```
ADDCOMSNMP   COM(ITSO)
  INTNETADR('8.6.5.4' '8.6.5.3')
  OBJACC(*WRITE)
```

This command adds the community ITSO to the SNMP agent community list. SNMP managers with internet addresses 8.6.5.4 and 8.6.5.3 are the only managers in the community and are able to change all MIB objects.

Figure 188 on page 270 shows the prompt screen for the ADDCOMSNMP command, which you can see on the AS/400 by typing the command ADDCOMSNMP at the command line and pressing PF4 the Prompt key.

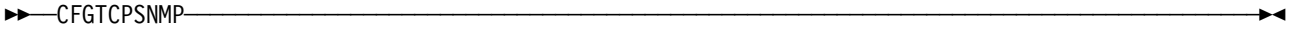
```
                    Add Community for SNMP (ADDCOMSNMP)

Type choices, press Enter.

Community name . . . . . .



Translate community name . . . .   *YES          *YES, *NO
Manager internet address . . . .   *ANY
             + for more values
Object access  . . . . . . . . .   *SNMPATR      *SNMPATR, *READ, *WRITE...
Log set requests . . . . . . . .   *SNMPATR      *SNMPATR, *YES, *NO
Log get requests . . . . . . . .   *SNMPATR      *SNMPATR, *YES, *NO









                                                                      Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 188. ADDCOMSNMP Prompt Panel

D.2 CFGTCPSNMP (Configure TCP/IP SNMP) Command

The format of the CFGTCPSNMP command is as follows:

Job: I Pgm: I REXX: I Exec



D.2.1 Purpose

The Configure TCP/IP SNMP (CFGTCPSNMP) command is used to display a menu that allows a user to define or change the Simple Network Management Protocol (SNMP) configuration. The menu options include:

- Change SNMP Attributes
- Work with Communities for SNMP

It is not necessary to run the CFGTCPSNMP command before using the SNMP agent. The SNMP agent is shipped with a community that has the following characteristics:

Community Name public

ASCIICOM *YES

INTNETADR *ANY

OBJACC *READ

LOGSET *NO

LOGGET *NO

See the Change SNMP Attributes (CHGSNMPA) command for the default values for SNMP attributes.

There are no parameters for this command. For example:

```
CFGTCPSNMP
```

Figure 189 on page 272 shows the Configure TCP/IP SNMP menu this command displays after typing CFGTCPSNMP on the AS/400 command line and pressing the Enter key.

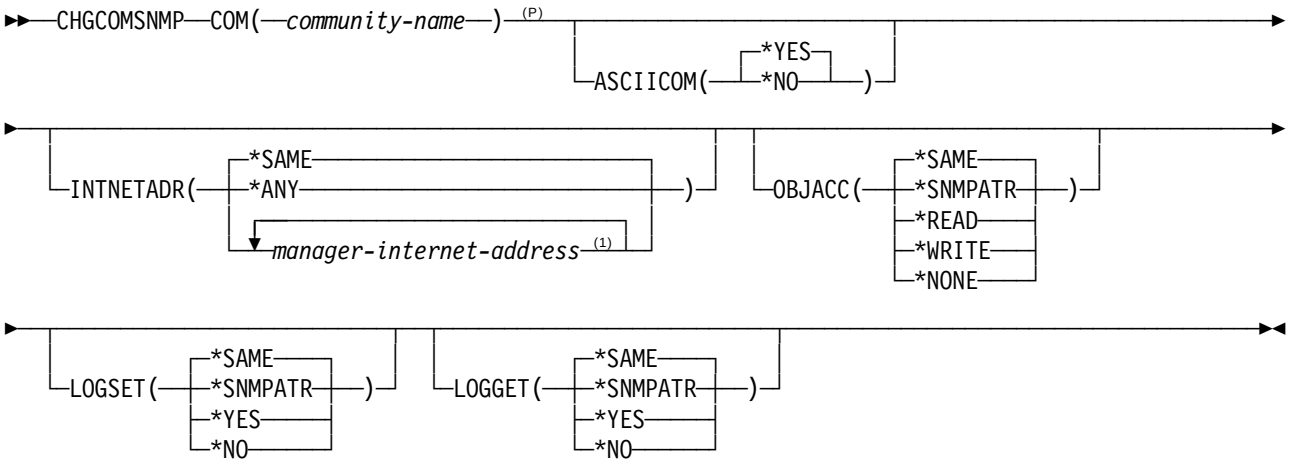
```
                              Configure TCP/IP SNMP
                                                              System:   RALYAS4B
Select one of the following:

     1. Change SNMP attributes
     2. Work with communities for SNMP









Selection or command
===>

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
(C) COPYRIGHT IBM CORP. 1980, 1994.
```

Figure 189. CFGTCPSNMP Menu Panel

D.3 CHGCOMSNMP (Change Community for SNMP) Command

The format of the CHGCOMSNMP command is as follows:

Job: B,I Pgm: B,I REXX: B,I Exec



Notes:

P All parameters preceding this point can be specified in positional form.

1 A maximum of 300 repetitions.

D.3.1 Purpose

The Change Community for SNMP (CHGCOMSNMP) command changes an SNMP community profile in the SNMP agent community list. An SNMP agent uses a community profile to determine whether or not to honor a request sent by an SNMP manager. The community profile consists of a community name, an object access specification, and a list of the SNMP managers that are part of the community. The combination of the community name (COM) and the translate to ASCII community (ASCIICOM) parameters defines a community.

D.3.2 Required Parameter

COM

This specifies the name of the SNMP community being changed. The community must already exist in the SNMP agent community list. You can define an SNMP community using the Add Community for SNMP (ADDCOMSNMP) command.

Community-name: Specify the name of the SNMP community being changed. The name may contain characters that cannot be displayed.

D.3.3 Optional Parameters

ASCIICOM

This specifies whether the community name is translated to ASCII characters before it is compared with the community name specified in a request from an SNMP manager. This parameter is used in combination with the community name to determine the community to be changed. If this parameter is not specified and two communities have the same name but

different ASCIICOM parameter values, the community that is changed is the community with ASCIICOM set to *YES.

***YES:** The community name is translated to ASCII characters before it is compared with a community name specified by an SNMP manager.

***NO:** The community name is not translated to ASCII characters before it is compared with a community name specified by an SNMP manager.

INTNETADR

These are the internet addresses of the SNMP managers that are part of this community.

***SAME:** The value does not change.

***ANY:** Allow any SNMP manager to be part of this community.

Manager-internet-address: Specify the internet address of the SNMP manager. The internet address is specified in the form *nnn.nnn.nnn.nnn*, where *nnn* is a decimal number ranging from 0 through 255. An internet address is not valid if it has a value of all binary ones or all binary zeros for the network identifier (ID) portion or the host ID portion of the address. If the internet address is entered from a command line, the address must be enclosed in apostrophes. Up to 300 unique internet addresses may be specified. The same internet address may appear in more than one community profile.

OBJACC

This specifies the object access for the community.

***SAME:** The value does not change.

***SNMPATR:** The object access defined with the Change SNMP Attributes (CHGSNMPA) command is used for this community.

***READ:** Allow SNMP managers that are part of this community to read all management information base (MIB) objects. Modification of MIB objects by SNMP managers is not permitted.

***WRITE:** Allow SNMP managers that are part of this community to change all MIB objects that can be changed. Specifying *WRITE implies *READ access.

***NONE:** Do not allow SNMP managers that are part of this community to access any MIB objects.

LOGSET

This specifies whether SET requests from SNMP managers in this community are logged in journal QSNMP in library QUSRSYS.

***SAME:** The value does not change.

***SNMPATR:** The value defined with the Change SNMP Attributes (CHGSNMPA) command is used for this community.

***YES:** SET requests are logged.

***NO:** SET requests are not logged.

LOGGET

This specifies whether GET requests and GETNEXT requests from SNMP managers in this community are logged in journal QSNMP in library QUSRSYS.

***SAME:** The value does not change.

***SNMPATR:** The value defined with the Change SNMP Attributes (CHGSNMPA) command is used for this community.

***YES:** GET requests and GETNEXT requests are logged.

***NO:** GET requests and GETNEXT requests are not logged.

An example of the use of the CHGCOMSNMP command follows:

```
CHGCOMSNMP   COM(RALEIGH)   INTNETADR(*ANY)
  OBJACC(*READ)
```

This command changes community RALEIGH to have an object access of read and to allow any SNMP manager in the community to read the MIB objects on this system. All of the other community values are unchanged.

On the AS/400 you can see the screen for the command CHGCOMSNMP as shown in Figure 190, by typing the command on the command line and pressing PF4 the Prompt key.

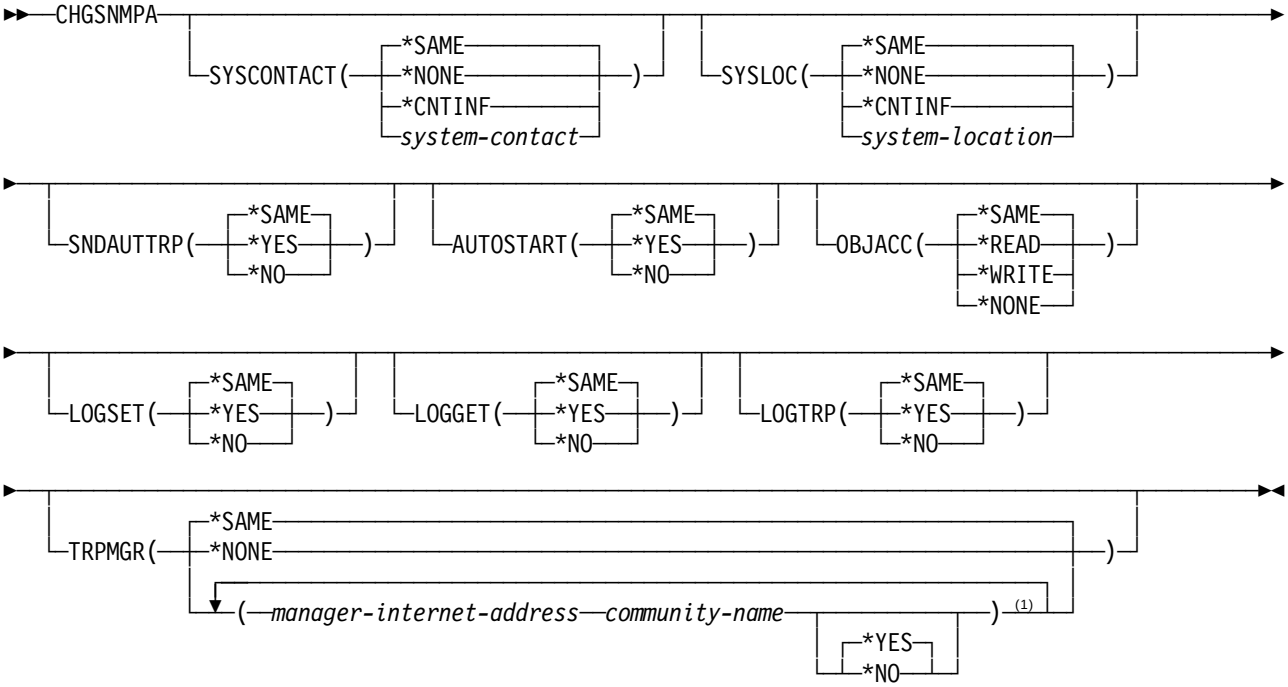
```
                     Change Community for SNMP (CHGCOMSNMP)

 Type choices, press Enter.

 Community name . . . . . .

 Translate community name . . . .   *YES          *YES, *NO

                                                                    Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

Figure 190. CHGCOMSNMP Prompt Panel

D.4 CHGSNMPA (Change SNMP Attributes) Command

The format of the CHGSNMPA command is as follows:

Job: B,I Pgm: B,I REXX: B,I Exec



Note:

[1] A maximum of 300 repetitions.

D.4.1 Purpose

The Change SNMP Attributes (CHGSNMPA) command changes values and options used by the OS/400 SNMP agent. The command is also used to specify which SNMP managers receive traps generated by the local AS/400 system.

The SNMP agent is shipped with the following values for the SNMP attributes.

Keyword	Value
SYSCONTACT	*NONE
SYSLOC	*NONE
SNDAUTTRP	*YES
AUTOSTART	*NO
OBJACC	*READ
LOGSET	*NO
LOGGET	*NO
LOGTRP	*NO
TRPMGR	*NONE

D.4.2 Optional Parameters

SYSCONTACT

This specifies the name of the contact person for this AS/400 system, along with information on how to contact this person. This value is used only by SNMP-specific functions. This value may be read or modified by an authorized SNMP manager.

***SAME:** The value does not change.

***NONE:** No system contact exists.

***CNTINF:** The value is obtained from the service contact information specified by using the Work with Contact Information (WRKCNTINF) command. The value obtained consists of the contact person and the contact telephone numbers.

System-contact: Specify the name of the contact person and other contact information. All of the characters specified must be able to be translated into the ASCII character set.

SYSLOC

This specifies the physical location of this AS/400 system. This value is used only by SNMP-specific functions. This value may be read or modified by an authorized SNMP manager.

***SAME:** The value does not change.

***NONE:** No system location information exists.

***CNTINF:** The value is obtained from the service contact information specified by using the Work with Contact Information (WRKCNTINF) command. The value obtained consists of the mailing address.

System-location: Specify the physical location of the system. All of the characters specified must be able to be translated into the ASCII character set.

SNDAUTTRP

This specifies whether the SNMP agent may send any authenticationFailure traps to any defined SNMP managers. An authenticationFailure trap is sent by the SNMP agent if a request is received from an SNMP manager that contains a community name that is not recognized by the SNMP agent. This trap is only sent when SNDAUTTRP is *YES and when at least one trap manager has been defined. This value may also be read or modified by an authorized SNMP manager.

***SAME:** The value does not change.

***YES:** authenticationFailure traps may be sent.

***NO:** authenticationFailure traps are not sent.

AUTOSTART

This specifies whether the SNMP agent is started when the STRTCP command runs.

***SAME:** The value does not change.

***YES:** The SNMP agent is started when the STRTCP command runs.

***NO:** The SNMP agent is not started when the STRTCP command runs.

OBJACC
This specifies the default object access for SNMP communities.

***SAME:** The value does not change.

***READ:** Allow SNMP managers that are part of a community to read all Management Information Base (MIB) objects. Modification of MIB objects by SNMP managers is not permitted.

***WRITE:** Allow SNMP managers that are part of a community to modify all MIB objects that can be modified. Specifying *WRITE implies *READ access.

***NONE:** Do not allow SNMP managers that are part of a community to modify any MIB objects.

LOGSET
This specifies the default value for whether SET requests from SNMP managers in a community are logged in journal QSNMP in library QUSRSYS.

***SAME:** The value does not change.

***YES:** SET requests are logged.

***NO:** SET requests are not logged.

LOGGET
This specifies the default value for whether GET requests and GETNEXT requests from SNMP managers in a community are logged in journal QSNMP in library QUSRSYS.

***SAME:** The value does not change.

***YES:** GET requests and GETNEXT requests are logged.

***NO:** GET requests and GETNEXT requests are not logged.

LOGTRP
This specifies whether traps are logged in journal QSNMP in library QUSRSYS.

***SAME:** The value does not change.

***YES:** Traps are logged.

***NO:** Traps are not logged.

TRPMGR
This specifies which SNMP managers receive traps generated by this AS/400 system.

***SAME:** The value does not change.

***NONE:** No SNMP managers receive traps.

Element 1: Manager internet Address

Manager-internet-address: Specify the internet address of the SNMP manager. The address must be of the form *nnn.nnn.nnn.nnn*, where *nnn* is a decimal number ranging from 0 to 255. This address is independent of the manager internet address specified on the ADDCOMSNMP and CHGCOMSNMP commands.

Element 2: Community Name

Community-name: Specify the SNMP community name to be placed in the traps sent to this SNMP manager. The community name specified in this element is independent of the community name specified on the ADDCOMSNMP,

CHGCOMSNMP, and RMVCOMSNMP commands. The name may contain characters that cannot be displayed.

Element 3: Translate Community Name

***YES:** The community name is translated to ASCII characters when a trap is sent to the SNMP manager. This value should be specified when the community name consists entirely of characters that can be displayed. An error message is sent if the community name cannot be translated to ASCII characters.

***NO:** The community name is not translated to ASCII characters when a trap is sent to the SNMP manager. This value should be specified when the community name contains one or more characters that cannot be displayed.

Following are two examples of the use of the CHGSNMPA command:

Example 1: Changing System Contact and Automatic Start

```
CHGSNMPA   SYSCONTACT('JOHN DOE, PHONE 123-4567')
  AUTOSTART(*NO)
```

This command changes the system contact information and specifies that the SNMP agent should not start when the STRTCP command runs. All other values are unchanged.

Example 2: Changing Trap Managers

```
CHGSNMPA   TRPMGR(('9.8.7.6' 'TRAPCOMMUNITY')
  ('9.8.7.5' 'TRAPCOMMUNITY2'))
```

This command causes any traps generated by the local AS/400 system to be sent to SNMP managers that have internet protocol addresses 9.8.7.6 and 9.8.7.5. Community name TRAPCOMMUNITY is placed in traps sent to 9.8.7.6, and community name TRAPCOMMUNITY2 is placed in traps sent to 9.8.7.5. For both managers the community name is translated to ASCII characters before being placed in the trap.

Figure 191 on page 280 shows the CHGSNMPA command prompt screen, which you can view by typing CHGSNMPA at the AS/400 command line and pressing PF4. Notice that the first prompt screen indicates that there is more information on the next screen.

```
                      Change SNMP Attributes (CHGSNMPA)

Type choices, press Enter.

System contact . . . . . .   *NONE


System location  . . . . . .   *NONE


Send authentication traps  . . .   *YES          *SAME, *YES, *NO
Automatic start  . . . . . .   *NO           *SAME, *YES, *NO
Object access  . . . . . .   *READ         *SAME, *READ, *WRITE, *NONE
Log set requests . . . . . .   *NO           *SAME, *YES, *NO
Log get requests . . . . . .   *NO           *SAME, *YES, *NO
Log traps  . . . . . .   *NO           *SAME, *YES, *NO


                                                                    More...
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 191. CHGSNMPA Prompt Panel Part 1

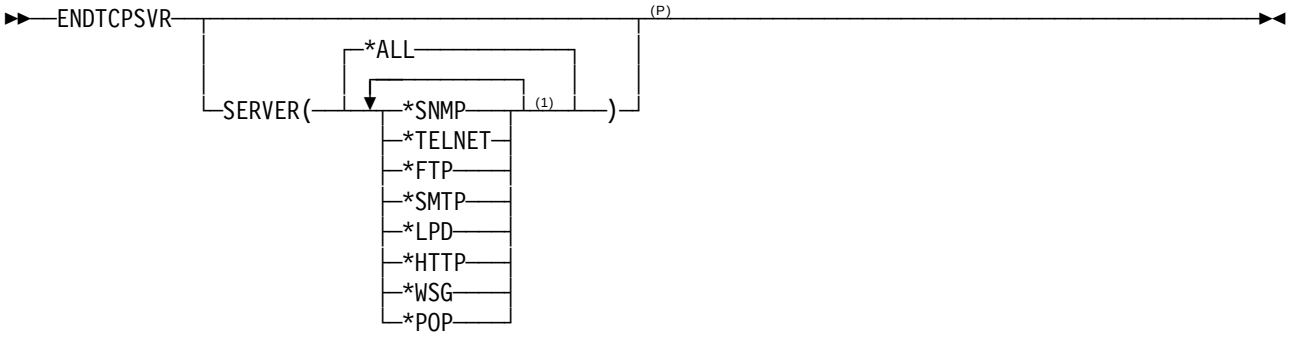
```
                      Change SNMP Attributes (CHGSNMPA)

Type choices, press Enter.

Trap managers:
  Manager internet address . . .   *NONE
  Community name . . . . . .


  Translate community name . . .   *YES          *YES, *NO
              + for more values









F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this dis
F24=More keys
```

Figure 192. CHGSNMPA Prompt Panel Part 2

D.5 ENDTCPSVR (End TCP/IP Server) Command

The format of the ENDTCPSVR command is as follows:

Job: B,I Pgm: B,I REXX: B,I Exec



Notes:

1 A maximum of 8 repetitions.

P All parameters preceding this point can be specified in positional form.

D.5.1 Purpose

The ENDTCPSVR command is used to end the TCP/IP application server jobs that are specified in the SERVER parameter. If the jobs have any current active connections, these connections are ended immediately. If the ENDTCPSVR command is used to end a server that is not active, a diagnostic message is returned.

D.5.2 Optional Parameter

SERVER

This specifies which of the TCP/IP application server jobs is to be ended by this command.

***ALL:** All of the TCP/IP server jobs are ended.

***SNMP:** All jobs associated with the SNMP agent in the QSYSWRK subsystem are ended.

***TELNET:** All TELNET server jobs are ended.

***FTP:** All FTP server jobs are ended.

***SMTP:** All jobs associated with SMTP in the QSYSWRK subsystem are ended. The bridge job in the QSNADS subsystem is not ended.

***LPD:** All LPD server jobs are ended.

***HTTP:** All World Wide Web HyperText Transfer Protocol server jobs are ended.

***WSG:** All HTML workstation gateway server jobs are ended.

***POP:** All Post Office Protocol (POP) Version 3 server jobs are ended.

Following are some examples of the use of the ENDTCPSVR command:

Example 1: Ending All TCP/IP Servers

```
ENDTCPSVR
```

This command ends all active TCP/IP application server jobs running in the QSYSWRK subsystem.

Example 2: Ending the LPD Servers

```
ENDTCPSVR   SERVER(*LPD)
```

This command ends the TCP/IP LPD application server jobs.

To view the ENDTCPSVR prompt screen shown in Figure 193, type the command at the AS/400 command line and press PF4.

```
                          End TCP/IP Server (ENDTCPSVR)

Type choices, press Enter.

Server application . . . . . .   *ALL          *ALL, *SNMP, *TELNET, *FTP...
               + for more values

                                                                        Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 193. ENDTCPSVR Command Prompt

If PF4 is pressed on this screen while the cursor is placed on the first prompt line, then the panel shown in Figure 194 on page 283 will be displayed.

```
                    Specify Value for Parameter SERVER

Type choice, press Enter.


Server application . . . . . .   *ALL

Single Values
  *ALL
Other Values
  *SNMP
  *TELNET
  *FTP
  *SMTP
  *LPD
  *HTTP
  *WSG
  *POP



F3=Exit   F5=Refresh   F12=Cancel   F13=How to use this display   F24=More keys
```

Figure 194. ENDTCPSVR Command Prompt SERVER Parameter Values

If a "+" symbol is entered on the prompt line, then the panel in Figure 195 will be displayed.

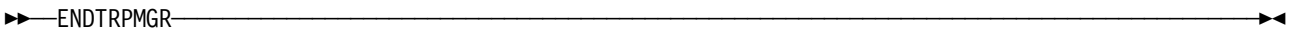
```
                  Specify More Values for Parameter SERVER

Type choices, press Enter.

Server application . . . . . .   *ALL          *ALL, *SNMP, *TELNET, *FTP...
                                 _____
                                 _____
                                 _____
                                 _____









                                                                        Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 195. ENDTCPSVR Command Prompt for Additional SERVER Parameter Values

D.6 ENDTRPMGR (End Trap Manager) Command

The format of the ENDTRPMGR command is as follows:

Job: B,I Pgm: B,I REXX: B,I Exec



D.6.1 Purpose

The End Trap Manager (ENDTRPMGR) command allows you to end the OS/400 SNMP Manager Framework trap manager job.

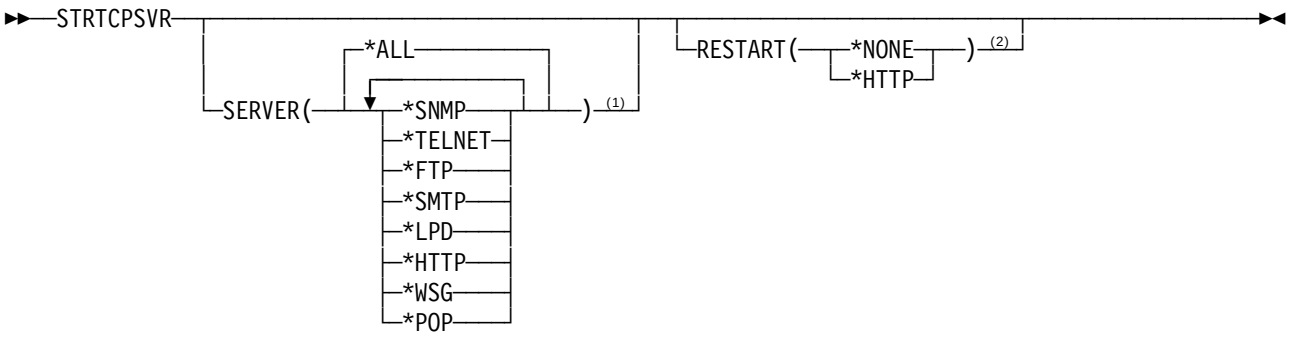
There are no parameters for this command.

There is no prompt screen for this command.

D.7 STRTCPSVR (Start TCP/IP Server) Command

The format of the STRTCPSVR command is as follows:

Job: B,I Pgm: B,I REXX: B,I Exec



Notes:

1 A maximum of 8 repetitions.

2 This parameter only takes effect if SERVER(*HTTP) is specified and HTTP server is running. Otherwise it will be ignored.

D.7.1 Purpose

The Start TCP/IP Server (STRTCPSVR) command is used to start the TCP/IP application servers that are shipped with the Operating System/400 product or the TCP/IP Connectivity Utilities/400 product. This command starts TCP/IP jobs in the QSYSWRK subsystem for the application servers specified with the server (SERVER) parameter. The number of server jobs started by this command is specified, where appropriate, in the configuration for each TCP/IP application.

All servers have an autostart (AUTOSTART) parameter on the associated configuration command, for example, Change FTP Attributes (CHGFTPA). This parameter indicates if the server should be started when the Start TCP/IP (STRTCP) command is entered. The STRTCPSVR command ignores the value of a server's autostart parameter.

D.7.2 Optional Parameter

SERVER

This specifies the TCP/IP application servers to be started by this command.

***ALL:** All of the TCP/IP application servers are started.

***SNMP:** The Simple Network Management Protocol (SNMP) agent jobs are started. Subsequent usage of the STRTCPSVR SERVER(*SNMP) command results in a diagnostic message if the SNMP server jobs have already been started.

***TELNET:** The TELNET server is started. Subsequent usage of the STRTCPSVR SERVER(*TELNET) command starts one additional TELNET server.

Note: Having more than one TELNET server job running reduces the chances of having connection attempts refused.

***FTP:** The File Transfer Protocol (FTP) servers are started based on the number of servers configured with the Change FTP Attributes (CHGFTPA) command. Subsequent usage of the STRTCPSVR SERVER(*FTP) command starts one additional FTP server.

Note: Having more than one FTP server job running can improve the performance of initiating a session when multiple users attempt to connect to the server in a short period of time.

***SMTP:** The Simple Mail Transfer Protocol (SMTP) client and server jobs are started. Additional SMTP client and server jobs cannot be started. Subsequent usage of the STRTCPSVR SERVER(*SMTP) command results in a diagnostic message if the SMTP server jobs have already been started.

***LPD:** The line printer daemon (LPD) servers are started based on the number of servers configured with the Change LPD Attributes (CHGLPDA) command. Subsequent usage of the STRTCPSVR SERVER(*LPD) command starts one additional LPD server.

***HTTP:** The World Wide Web Hyper Text Transfer Protocol (HTTP) servers are started based on the number of servers specified in CHGHTTPA command. Subsequent use of STRTCPSVR SERVER(*HTTP) results in a diagnostic message if the HTTP server has already been started.

***WSG:** The HTML WSG server is started. Subsequent use of STRTCPSVR SERVER(*WSG) results in a diagnostic message if the WSG server has already been started.

***POP:** The Post Office (POP) Version 3 servers are started, based on the number of servers configured with CHGPOPA command. Subsequent use of STRTCPSVR SERVER(*POP) results in a diagnostic message if the POP server has already been started.

Note: LPD works most efficiently when two or more servers are running. Running only one server works, but no jobs can be received while a current job is running. If a large print job is running, new jobs have to wait before LPD is ready to accept any new line printer requester (LPR) requests.

RESTART
Specifies whether to restart the selected server, when STRTCPSVR is issued. The SERVER parameter specified must be *ALL or *HTTP, or this parameter is ignored (at this time only *HTTP server is supported with the RESTART parameter).

Following are two examples of the use of the STRTCPSVR command:

Example 1: Starting all TCP/IP Servers

```
STRTCPSVR
```

This command starts all of the TCP/IP application servers that have been configured to be started. For example, if the Change FTP Attributes (CHGFTPA) command had previously been used to configure two FTP servers, both servers would be started when STRTCPSVR is issued. This example is also true for other TCP/IP application servers.

Where appropriate, the number of servers to start is based on the number of servers configured for the server being started. The configuration option to automatically start the servers (AUTOSTART) is ignored by the STRTCPSVR command. The AUTOSTART parameter is used only by the STRTCP command.

Example 2: Starting the TELNET Server

```
STRTCPSVR  SERVER(*TELNET)
```

This command starts the TCP/IP TELNET application server. If the TELNET server had been previously started, one additional TELNET server job would be started.

Figure 196 on page 287 shows the prompt screen for the STRTCPSVR command. To view this panel on your AS/400, type STRTCPSVR on the command AS/400 line and press PF4.

```
                        Start TCP/IP Server (STRTCPSVR)

Type choices, press Enter.

Server application . . . . . .   *ALL          *ALL, *SNMP, *TELNET, *FTP...
               + for more values

                                                                        Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 196. STRTCPSVR Command Prompt

If PF4 is pressed on this screen while the cursor is placed on the first prompt line, then the panel shown in Figure 197 will be displayed.

```
                   Specify Value for Parameter SERVER

 Type choice, press Enter.


 Server application . . . . . .   *ALL

 Single Values
   *ALL
 Other Values
   *SNMP
   *TELNET
   *FTP
   *SMTP
   *LPD
   *HTTP
   *WSG
   *POP



 F3=Exit   F5=Refresh   F12=Cancel   F13=How to use this display   F24=More keys
```

Figure 197. STRTCPSVR Command Prompt for SERVER Parameter Values

If a "+" symbol is entered on the prompt line, then the panel in Figure 198 on page 288 will be displayed.

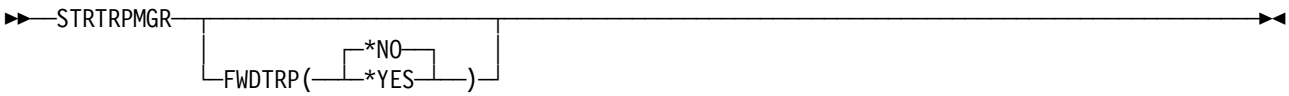
```
                    Specify More Values for Parameter SERVER

 Type choices, press Enter.

 Server application . . . . . .   *ALL          *ALL, *SNMP, *TELNET, *FTP...
                                   _______
                                   _______
                                   _______
                                   _______

                                                                        Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
```

Figure 198. STRTCPSVR Command Prompt for Additional SERVER Parameter Values

D.8 STRTRPMGR (Start Trap Manager) Command

The format of the STRTRPMGR command is as follows:

Job: B,I Pgm: B,I REXX: B,I Exec



D.8.1 Purpose

The Start Trap Manager (STRTRPMGR) command allows you to start the OS/400 SNMP Manager Framework trap manager job. An optional Forward Trap parameter may be specified which enables traps that are received from other systems to be forwarded to other Network Management stations. The trap manager uses the trap generation and sending facilities provided in the Simple Network Management Protocol (SNMP) agent and Distributed Protocol Interface (DPI).

D.8.2 Optional Parameters

FWDTRP

This specifies whether traps received on the system are to be forwarded to other network management stations.

***YES:** Received traps are forwarded using the facilities provided in the SNMP agent and DPI interface.

***NO:** Received traps are not forwarded. Traps are only enqueued.

Following are two examples of the use of the STRTRPMGR command:

Example 1: Start Trap Manager Job (Enqueue Traps Only)

```
STRTRPMGR
```

This command starts the trap manager job. Traps received by the trap manager are enqueued only.

Example 2: Start Trap Manager Job (Enqueue & Forward Traps)

```
STRTRPMGR FWDTRP(*YES)
```

This command starts the trap manager job. Traps received by the trap manager are enqueued and forwarded.

Figure 199 on page 290 shows the prompt screen for the STRTRPMGR command. To view this panel on your AS/400, type STRTRPMGR on the AS/400 command line and press PF4.

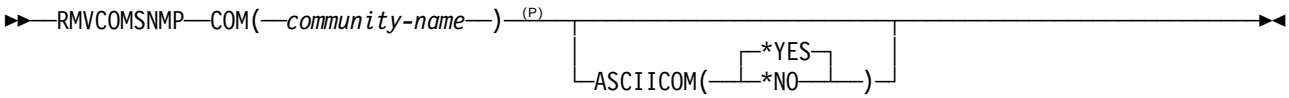
```
                        Start Trap Manager (STRTRPMGR)

Type choices, press Enter.

Forward Traps  . . . . . .   *NO           *YES, *NO
                                                                        Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 199. STRTRPMGR Command Prompt

D.9 RMVCOMSNMP(Remove Community for SNMP) Command

The format of the RMVCOMSNMP command is as follows:

Job: B,I Pgm: B,I REXX: B,I Exec



Note:
[P] All parameters preceding this point can be specified in positional form.

D.9.1 Purpose

The Remove Community for SNMP (RMVCOMSNMP) command is used to remove a Simple Network Management Protocol (SNMP) community profile from the SNMP agent community list. The community profile consists of a community name, an object access specification, and a list of the SNMP managers that are part of the community. The combination of the community name (COM) and the translate to ASCII community (ASCIICOM) parameters defines a community.

D.9.2 Required Parameter

COM

This specifies the name of the SNMP community being removed. The community must already exist in the SNMP agent community list.

Community-name: Specify the name of the SNMP community being removed. The name may contain characters that cannot be displayed.

D.9.3 Optional Parameter

ASCIICOM

This specifies whether the community name is translated to ASCII characters before it is compared with the community name specified in a request from an SNMP manager. This parameter is used in combination with the community name to determine the community to be removed. If two communities have the same name and you don't specify the ASCIICOM parameter, the community with ASCIICOM set to *YES is removed.

***YES:** The community name is translated to ASCII characters before it is compared with a community name specified by an SNMP manager.

***NO:** The community name is not translated to ASCII characters before it is compared with a community name specified by an SNMP manager.

Following is an example of the use of the RMVCOMSNMP command:

```
RMVCOMSNMP   COM(OCOEE)
```

This command removes community OCOEE from the SNMP agent community list.

If you wish to view the prompt screen for the RMVCOMSNMP command, type the command on the AS/400 command line and press PF4. The resulting screen is shown in Figure 200 on page 292.

```
                    Remove Community for SNMP (RMVCOMSNMP)

Type choices, press Enter.

Community name . . . . . .

Translate community name . . . .   *YES          *YES, *NO

                                                                    Bottom
F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
F24=More keys
```

Figure 200. RMVCOMSNMP Prompt Panel

Appendix E. Customizing the NetView for OS/2 Discovery Process

In Chapter 5, “NetView for OS/2” on page 41 we talked about the NetView for OS/2 discovery process. In this appendix we will explain how to control this discovery process using a combination of seed files and mask files.

IP discovery is the management program that attempts to discover systems on the network that have an IP address. After NetView for OS/2 discovers a system, it polls the system regularly to check for updated status information. The two ways of customizing IP discovery are:

- Changing the scope of the initial discovery by using a seed file.
- Limiting the systems added to the discovery database by the use of a mask file.

E.1 Creating a Seed File

A seed file is used to ensure NetView for OS/2 attempts to discover certain systems. Without a seed file, NetView for OS/2 will discover things as and when it detects IP traffic flowing on the network. It obviously makes more sense for your managing system to discover all the systems that need to be managed as quickly as possible.

In the following example we will create a seed file to discover the two AS/400s, RALYAS4A and RALYAS4B.

To create a seed file you need to edit a NetView for OS/2 file called SEEDLIST.LOG in the \anv2\etc directory; in our case the path was D:\anv2\etc\seedlist.log.

Edit the file to include those systems you wish to discover at startup. You are allowed to use a # to comment out unwanted addresses. See Figure 201 for our resultant seed file.

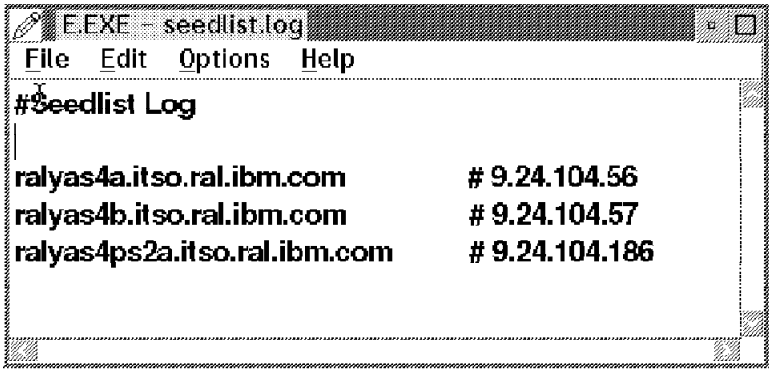


Figure 201. Our Completed Seed File, SEEDLIST.LOG

For the discovery process to start using this seed file we have to edit the LNTCPIP.LRF file. The path for this on our system is D:\anv2\etc\lrf\lntcpip.lrf.

You should edit the file such that it reads as follows:

```
tcpipdiscovery:LNTCPIP.EXE:
OVs_YES_START:postmaster,toplogy,agentdiscovery:-s anv2\etc\seedlist.log,-L:OVs_DAEMON::
```

As shown in Figure 202.

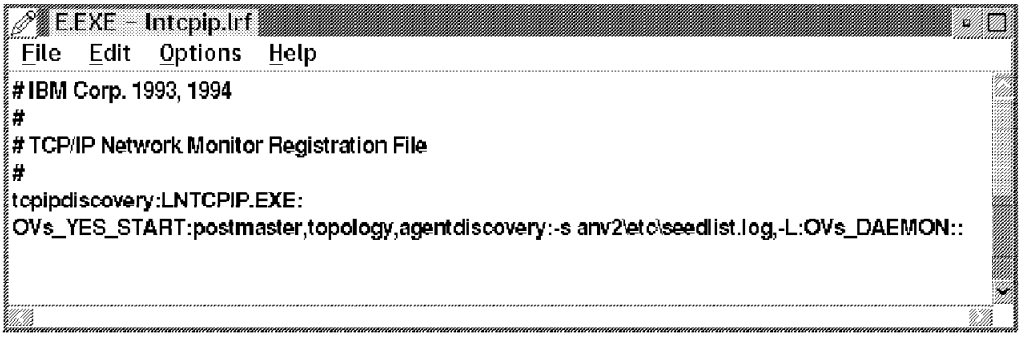


Figure 202. Editing the LNTCPIP.LRF File

When you have finished editing the file, save it back to the LRF directory.

Stop TCP/IP discovery by typing:

```
SVSTOP tcpipdiscovery
```

Add the lntcpip.lrf file by typing:

```
SVADDOBJ LNTCPIP.LRF
```

You should get a message that indicates that it completed successfully.

Start TCP/IP discovery by typing:

```
SVSTART tcpipdiscovery
```

E.2 Creating a Mask File

Now that we have created a seed file to ensure we discover our managed systems, we will create a mask file. This will add discovered systems to the All Systems folder according to our specified criteria.

Without a mask file, the All Systems folder will become populated with all the IP systems in your network and become unmanageable. See Figure 203 on page 295 for an example of our All Systems folder before we started using a mask file.

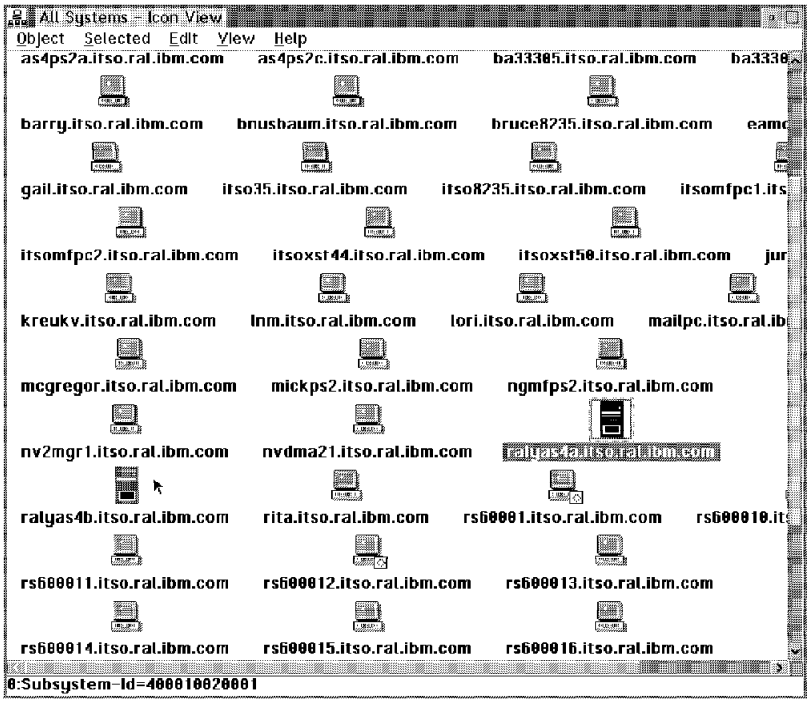


Figure 203. All Systems Folder without Using Mask File

To create a mask file you will need to edit file LNTCPIP.MSK. On our system it is found in path D:\anv2\etc\lntcpip.msk. Once changed, the new mask will take effect, however, all systems already discovered will remain in the discovery database.

The default mask is *.*.*.*. That is, all systems are added to the folder.

We changed the mask as follows:

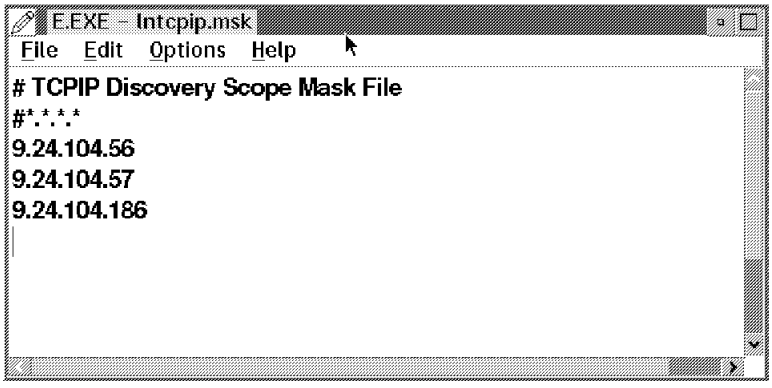


Figure 204. Finished Mask File

This limits the discovery process to systems RALYAS4A, RALYAS4B and AS4PS2A. The SNMP manager, AS4PS2A, will also discover itself by default. When we reboot the PC, the All Systems folder is cleared and the discovery process starts again, this time using the parameters defined in the seed file and mask file. See Figure 205 on page 296 for our All Systems folder when using seed and mask files.

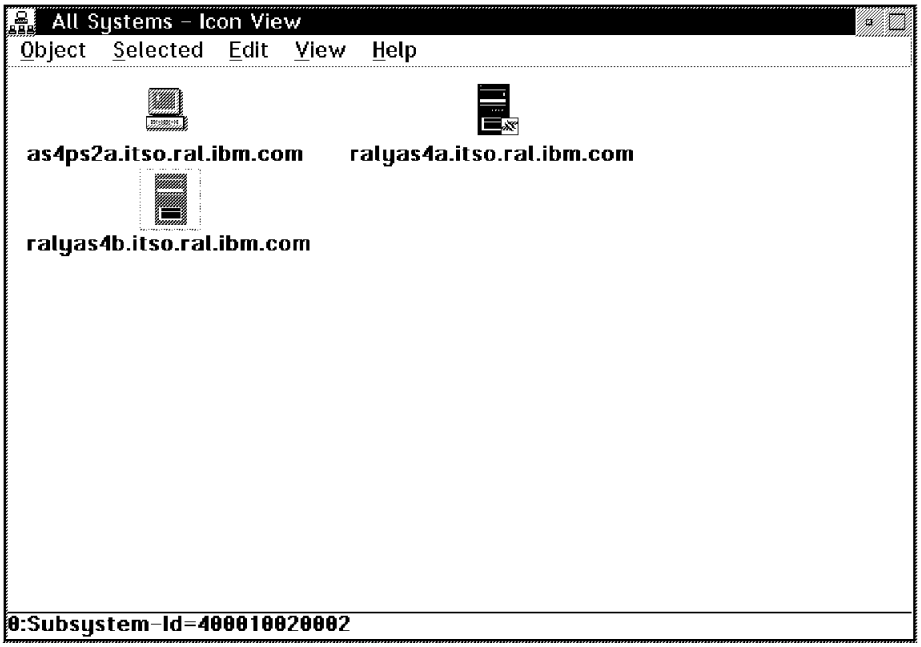


Figure 205. Three Systems Discovered

Appendix F. SNMPv2

SNMPv2 is a new version of SNMP; it is documented in twelve RFCs. SNMPv2 was developed in order to give a better response to security and operational problems.

Up-to-date SNMPv2 information can be obtained by accessing the following world wide web site:

```
http://www.snmp.com/v2star.html
```

F.1 Security

In the original SNMP, the administrative relationship between an agent and one or more management applications was identified by a community. The community relationship involved the following three aspects:

- Identification of the entities authorized to request management operations
- Identification of the type of management operation that is allowed (read, write or none)
- Identification of management information that is available to the operations (MIB views)

  Note: MIB views are not implemented in OS/400 SNMP support.

Now with SNMPv2, three new concepts appear:

- The *party* concept which is an execution environment residing in an agent or management application, which refers to entities that communicate via a management protocol and a transport service using authentication and privacy facilities.
- The *context* concept refers to collections of managed objects resources accessible by an SNMPv2 entity.
- The *access policy* concept defines the operations that may be performed when a source party communicates with a destination party and references a particular context. There are three levels of authentication/protection:

  snmpPrivMsg contains the party name and an snmpAuthMsg the content of which is encrypted by secret key.

  snmpAuthMsg contains authentication credentials and information about the management operation and its execution environment.

  snmpMgtCom contains the name of the party that originated the message, the party that is intended to receive the message, the managed objects, and the desired operation.

F.2 Operational Model

Some of the operations of SNMP remained the same and some were added. The following is a list of the operations available in SNMPv2:

- GET: This operation experienced no change.
- GETNEXT: This operation experienced no change.

- SET: This operation experienced no change.
- GETBULK: This operator was introduced to minimize network interactions, by allowing the agent to return large packets. This operation gets everything under the MIB. The number of variables which should be retrieved (non-repeaters) and the maximum number of times that other variables should be retrieved (max-repetitions) can be specified. If *non-repeaters* is greater than or equal to the number of variables in the request or *non-repeaters* is equal to zero and *max-repetitions* equal to one, a GETNEXT operation would be being emulated.
- INFORM: This operator is used when a management application wishes to inform another management application of some information. This operation always receives a response from the other management application.

F.3 SNMPv2 RFCs

The new SNMPv2 framework is defined in the following twelve RFCs:

RFC1441 Introduction to SNMPv2

RFC1442 SMI for SNMPv2

RFC1443 Textual Conventions for SNMPv2

RFC1444 Conformance Statements for SNMPv2

RFC1445 Administrative Model for SNMPv2

RFC1446 Security Protocols for SNMPv2

RFC1447 Party MIB for SNMPv2

RFC1448 Protocol Operations for SNMPv2

RFC1449 Transport Mappings for SNMPv2

RFC1450 MIB for SNMPv2

RFC1451 Manager-to-Manager MIB

RFC1452 Coexistence between SNMPv1 and SNMPv2

For more information on how to request RFCs refer to B.1.1, “Request for Comments (RFC)” on page 246.

Appendix G. Special Notices

This publication is intended to help AS/400 specialists understand SNMP (Simple Network Management Protocol) and the AS/400′s implementation of SNMP. The book will also be useful to the non-AS/400 networking specialist who is looking for information on the AS/400′s implementation of SNMP. The information in this publication is not intended as the specification of any programming interfaces that are provided by Operating System/400. See the PUBLICATIONS section of the IBM Programming Announcement for Operating System/400 for more information about what publications are considered to be product documentation.

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

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The 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:

AIX
APPN
AT
Client Access/400
IBM
Operating System/400
OS/400
400
AnyNet
AS/400
C/400
Client Access
NetView
OS/2
RS/6000

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

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

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

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

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

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

Appendix H. Related Publications

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

H.1 International Technical Support Organization Publications

- *NetView for OS/2 as an SNMP Manager*, GG24-4412
- *Inside Client Access/400 Optimized for OS/2*, SG24-2587

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 303.

H.2 Redbooks on CD-ROMs

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

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

H.3 Other Publications

These publications are also relevant as further information sources:

- *OS/400 Network Systems Management V3*, SC41-3409
- *NetView for OS/2 V2R1 Installation and Administration*, SC31-8100
- *NetView for OS/2 V2R1 User's Guide*, SC31-8099
- *Simple Network Management Protocol (SNMP) Support V3R2*, SC41-3412
- *Simple Network Management Protocol (SNMP) Support V3R6*, SC41-4412
- *OS/400 System API Reference: UNIX Type APIs V3R2*, SC41-3875 (available only on AS/400 Softcopy Library CD-ROM, SK2T-2171-05)
- *OS/400 System API Reference: UNIX Type APIs V3R6*, SC41-4875 (available only on AS/400 Softcopy Library CD-ROM, SK2T-2839)
- *TCP/IP Configuration And Reference Book V3*, SC41-3420

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

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

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type one of the following commands:

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

  To register for information on workshops, residencies, and redbooks, type the following command:

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

  For a list of product area specialists in the ITSO: type the following command:

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

- **Redbooks Web Site on the World Wide Web**

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

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

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.htm). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

How Customers Can Get ITSO Redbooks

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

- **Online Orders** — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

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

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

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

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.htm). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

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

Index

A

Abstract Syntax Notation.1 (ASN.1) 15
Access mode 239
Access policy 239
ADDCOMSNMP (Add Community for SNMP) Command 267
Advanced Peer-To-Peer Networking (APPN) MIB 26, 159
Agent 6, 12, 24
AnyNet 227
Application entities 238
Architecture 9
AS/400 as a trap generator 205
ASN.1 notation 16
Authentic SNMP message 238
Authentication scheme 238
Authentication service 238
authenticationFailure trap 244

B

bibliography 301

C

CFGTCPSNMP (Configure TCP/IP SNMP) Command 271
CHGCOMSNMP (Change Community for SNMP) Command 273
CHGSNMPA (Change SNMP Attributes) Command 276
Client Access Optimized for OS/2 197
Client Management MIB 26, 137, 170
CMIP over TCP (CMOT) 4
coldStart trap 244
Common Management Information Protocol (CMIP) 2, 4
Common Management Information Service (CMIS) 4
Community 6, 238
Community name 238
Community profile 239
Connection-Oriented 3
Connectionless 3
connectSNMP() 104, 106

D

debugDPI() 106
disconnectSNMP() 105, 106
Distributed Protocol Interface (DPI) 91
DPI 2.0 MIB 27, 158
DPI REGISTER 103
DPI requests
  DPI Close 102
DPI requests *(continued)*
  DPI Commit 99, 102
  DPI Connect 96
  DPI Get 97
  DPI Getnext 98
  DPI Open 96
  DPI Response 97
  DPI Set 98
  DPI Undo 99, 102
  DPI Unregister 102
DPI_PACKET_LEN() 106
Draft Standard Protocol 248

E

egpNeighborLoss trap 244
Elective Protocol 248
ENDTCPSVR (End TCP/IP Server) Command 281
ENDTRPMGR (End Trap Manager) Command 284
Enteprise-specific MIBs 14, 159
enterpriseSpecific trap 244
Ethernet-like Interface MIB 24, 150
Experimental MIBs 14
Experimental Protocol 248

F

FDDI MIB 25, 153
fDPIparse() 106
fDPIset() 106
Frame Relay MIB 25, 154

G

GET 20
GET-RESPONSE 21
GETNEXT 20
GetNextRequest-PDU 242
GetRequest-PDU 241
GetResponse-PDU 243

H

Heterogeneous Network 6
Historic Protocol 248
Host Resources MIB 150, 189
Host Resources MIB and AS/400 189
Host Resources MIB and Client Access 197

I

IBM Enterprise MIBs 26
IBMAPPN 219
IBMCLTM 219

IBMNV6SA 219
Instance 6
Internet Activities Board (IAB) 245
Internet Assigned Numbers Authority (IANA) 246
Internet Engineering Steering Group (IESG) 245
Internet Engineering Task Force (IETF) 245
Internet Research Steering Group (IRSG) 245
Internet Research Task Force (IRTF) 245
Internetwork Packet Exchange MIB 26
IPX MIB 26, 178

L

Leaf node 16
Limited Use Protocol 248
linkDown trap 244
linkUp trap 244

M

Management Information Base (MIB) 6
Manager 6, 13, 27, 75
Manager APIs 75
Mask file 294
Messages 240
MIB Object 6
MIB view 238
MIB-II 24, 148
mkDPIAreYouThere() 106
mkDPIclose() 103, 106
mkDPIopen() 103, 106
mkDPIregister() 106
mkDPIresponse() 98, 106
mkDPIset() 102, 106
mkDPItrap() 104, 106
mkDPIunregister() 106
Model 10

N

netbiosdiscovery 44
NetView for AIX 231
NetView for AIX Subagent MIB 171
NetView for OS/2 41
NetView/6000 Subagent MIB 26
Netware Link Services Protocol MIB 26
netwdiscovery 44
Network element (NE) 10
Network management station (NMS) 10
NLSP MIB 26, 187
Not Recommended Protocol 248
Novell Enterprise MIBs 26, 178

O

Object Instance 18
Object-identifier 7, 16, 18
Object-type 15, 18
OSI 3
  CMIP over TCP (CMOT) 4
  Common Management Information Protocol (CMIP) 4
  Common Management Information Service (CMIS) 4
  Protocol 3
  Service 3
  Service Access Point (SAP) 3

P

pDPIpacket() 98, 106
Problem Determination 251
Proposed Standard Protocol 248
Protocol 3
Protocol Data Unit 10, 241
Protocol entities 238
Proxy access policy 239
Proxy Agent 6, 239

Q

QANMMIB 219
QSNMP Journal 211

R

receiveDPIpacket() 105, 106
Recommended Protocol 248
Remote Workstation MIB 26, 171
Request For Comments (RFC) 7, 246
Request/Response Protocol 7, 11
Required Protocol 248
RFC1155 3, 16, 23, 172, 178, 189
RFC1157 3, 23
RFC1212 3, 23, 172, 178, 189
RFC1213 3, 23, 24, 147, 149, 172, 178, 189
RFC1215 23, 178
RFC1231 25, 147, 155
RFC1280 23
RFC1285 25, 147, 153
RFC1315 25, 147, 154
RFC1340 23
RFC1398 24, 147, 151
RFC1441 298
RFC1442 298
RFC1443 298
RFC1444 298
RFC1445 298
RFC1446 298
RFC1447 298
RFC1448 298
RFC1449 298
RFC1450 298
RFC1451 298
RFC1452 298
RFC1514 25, 189

RFC1592 27, 147, 158
RFC1593 160
RIP/SAP MIB 26, 186
RMVCOMSNMP(Remove Community for SNMP) Command 291
Routing Information and Service Advertising Protocols MIB 26

S

SAP MIB 186
Scalar (non-tabular) Objects 20
Seed File 293
sendDPIpacket() 98, 106
Service 3
Service Access Point (SAP) 3
SET 21
SetRequest-PDU 243
Simple Gateway Monitoring Protocol (SGMP) 2
SNMP Agent 6, 12, 24
SNMP Architecture 9
SNMP Community 6, 238
SNMP Manager 6, 13, 27, 75
SNMP Manager APIs 75
SNMP Messages 240
SNMP Model 10
SNMP Proxy Agent 6
SNMP Subagent 6, 13, 28, 91
SNMP Subagent MIB 27, 158
SNMP Trap 7
snmpGet 76
snmpGetnext 76
snmpSet 76
SNMPv2 297
Standard Protocol 248
Standard RFC MIBs 14, 24, 148
STRTCPSVR (Start TCP/IP Server) Command 285
STRTRPMGR (Start Trap Manager) Command 289
Structure of Management Information (SMI) 16
Subagent 6, 13, 28, 91
Subagent APIs 105
Subagent MIB 27, 158
Subtree 16
SVSTART 43
SVSTATUS 44
SVSTOP 45

T

Tabular Objects 19
tcpipdiscovery 44
The Host Resources MIB subtree 189
Token-Ring MIB 25, 155
Transport address 240
TRAP 21
Trap Manager 78
Trap Support 27
Trap-PDU 243

U

User Datagram Protocol (UDP) 10

V

VarBindList 239
Variable binding (VarBind) 239
Variable name 18

W

waitDPIpacket() 105, 106
warmStart trap 244

ITSO Redbook Evaluation

Introduction to IBM AS/400 SNMP Support
SG24-4504-01

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@vnet.ibm.com

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

_____

_____

_____

_____

_____

This soft copy for use by IBM employees only.

Printed in U.S.A.